

8

13/5/2015

v. 2.5.1

Interrupts

Interrupts can be seen as a number of functions. These functions make the programming much easier, instead of writing a code to print a character you can simply call the interrupt and it will do everything for you.

```
mov al, 'S'
mov ah, 0eh
int 10h
```

registers

	H	L
AX	0E	53
BX	00	00
CX	00	00
DX	00	00



8088 and 8086 interrupts:



- ✓ External Hardware Interrupts
- ✓ Nonmaskable Interrupt
- ✓ Software Interrupts
- ✓ Internal Interrupts
- ✓ Reset

INTRO TO INTERRUPTS

- An interrupt is a **hardware-generated CALL**
 - externally derived from a hardware signal
- Or a **software-generated CALL**
 - internally derived from the execution of an instruction or by some other internal event
 - at times an internal interrupt is called an *exception*
- Either type interrupts the program by calling an **interrupt service procedure (ISP)** or interrupt handler.

• Note

1. The starting address of an ISR in the 1Kb memory space is as the Interrupt Pointer or interrupt vector corresponding to that interrupt.

54

00 → FFH 0 → 255d ⇒ 4 × 64 / 255
→ 0 → 1023d

- An interrupt is an external event which informs the CPU that a device needs service
- In the 8088 & 8086 there are a total of 256 interrupts (or interrupt types)
 - INT 00
 - INT 01
 - ...
 - INT FF
- When an interrupt is executed, the microprocessor automatically saves the flags register (FR), the instruction pointer (IP) and the code segment register (CS) on the stack and goes to a fixed memory location.
- In 80x86, the memory location to which an interrupt goes is always four times the value of the interrupt number
- INT 03h goes to 000Ch

$$\begin{array}{r} 03 \\ \times 4 \\ \hline 0C \end{array}$$

000C } → IP
000D }
000E } → CS
000F }

Interrupt Vectors

- A 4-byte number stored in the first 1024 bytes of memory (00000H–003FFH) in real mode.
- 256 different interrupt vectors.
 - each vector contains the address of an interrupt service procedure
- Each vector contains a value for IP and CS that forms the address of the interrupt service procedure.
 - the first 2 bytes contain IP; the last 2 bytes CS

INTs

- 256 different software interrupt instructions (INTs) available to the programmer.
 - each INT instruction has a numeric operand whose range is 0 to 255 (00H–FFH)
- For example, INT 100 uses interrupt vector 100, which appears at memory address 190H–193H.
 - INT 10H instruction calls the interrupt service procedure whose address is stored beginning at memory location 40H ($10H \times 4$) in the mode
- Each INT instruction is 2 bytes long.
 - the first byte contains the opcode
 - the second byte contains the vector type number
- When a software interrupt executes, it:
 - pushes the flags onto the stack
 - clears the T and I flag bits
 - pushes CS onto the stack
 - fetches the new value for CS from the interrupt vector
 - pushes IP onto the stack
 - fetches the new value for IP/EIP from the vector
 - jumps to the new location addressed by CS and IP

$100d = 64H$
INT 64H
 $\frac{64}{4}$
190H

Interrupt Service Routine

- For every interrupt, there must be a program associated with it
- This program is called an Interrupt Service Routine (ISR)
- It is also called an interrupt handler
- When an interrupt occurs, CPU runs the interrupt handler but where is the handler?
 - In the interrupt Vector Table (IVT)

INT Number	Physical Address	Contains
INT 00	00000h	IP0:CS0
INT 01	00004h	IP1:CS1
INT 02	00008h	IP2:CS2
.	.	.
INT FF	003FCh	IP255:CS255

Examples

For example: vector 50: CS and IP?

Physical Address $200 = (4 \times 50) = 200 = 11001000 = C8H$

000C8 contains IP: and 000CA contains CS information

- INT 12 (or vector 12)
- The physical address $30h (4 \times 12 = 48 = 30h)$ contains
 - 0030h and 0031h contain IP of the ISR
 - 0032h and 0033h contain CS of the ISR

IRET

- Used only with software or hardware interrupt service procedures.
- IRET instruction will
 - pop stack data back into the IP
 - pop stack data back into CS
 - pop stack data back into the flag register

Differences between INT and CALL

- ❖ A CALL FAR instruction can jump any location within the 1 MB address range but INT nn goes to a fixed memory location in the Interrupt Vector Table to get the address of the interrupt service routine
- ❖ A CALL FAR instruction is used by the programmer in the sequence of instruction in the program but externally activated hardware interrupt can come at any time
- ❖ A CALL FAR cannot be masked but INT nn in hardware can be blocked.
- ❖ A CALL FAR saves CS:IP but INT nn saves Flags and CS:IP
- ❖ At the end of the subroutine RET is used whereas for Interrupt routine IRET should be the last statement

Interrupt Instructions

Mnemonic	Meaning	Format	Operation	Flags Affected
CLI	Clear interrupt flag	CLI	$0 \rightarrow (IF)$	IF
STI	Set interrupt flag	STI	$1 \rightarrow (IF)$	IF
INT n	Type n software interrupt	INT n	$(Flags) \rightarrow ((SP) - 2)$ $0 \rightarrow TF, IF$ $(CS) \rightarrow ((SP) - 4)$ $(2 + 4 \cdot n) \rightarrow (CS)$ $(IP) \rightarrow ((SP) - 6)$ $(4 \cdot n) \rightarrow (IP)$	TF, IF
IRET	Interrupt return	IRET	$((SP)) \rightarrow (IP)$ $((SP) + 2) \rightarrow (CS)$ $((SP) + 4) \rightarrow (Flags)$ $(SP) + 6 \rightarrow (SP)$	All
INTO	Interrupt on overflow	INTO	INT 4 steps	TF, IF
HLT	Halt	HLT	Wait for an external interrupt or reset to occur	None
WAIT	Wait	WAIT	Wait for TEST input to go active	None

The Operation of Real Mode Interrupt

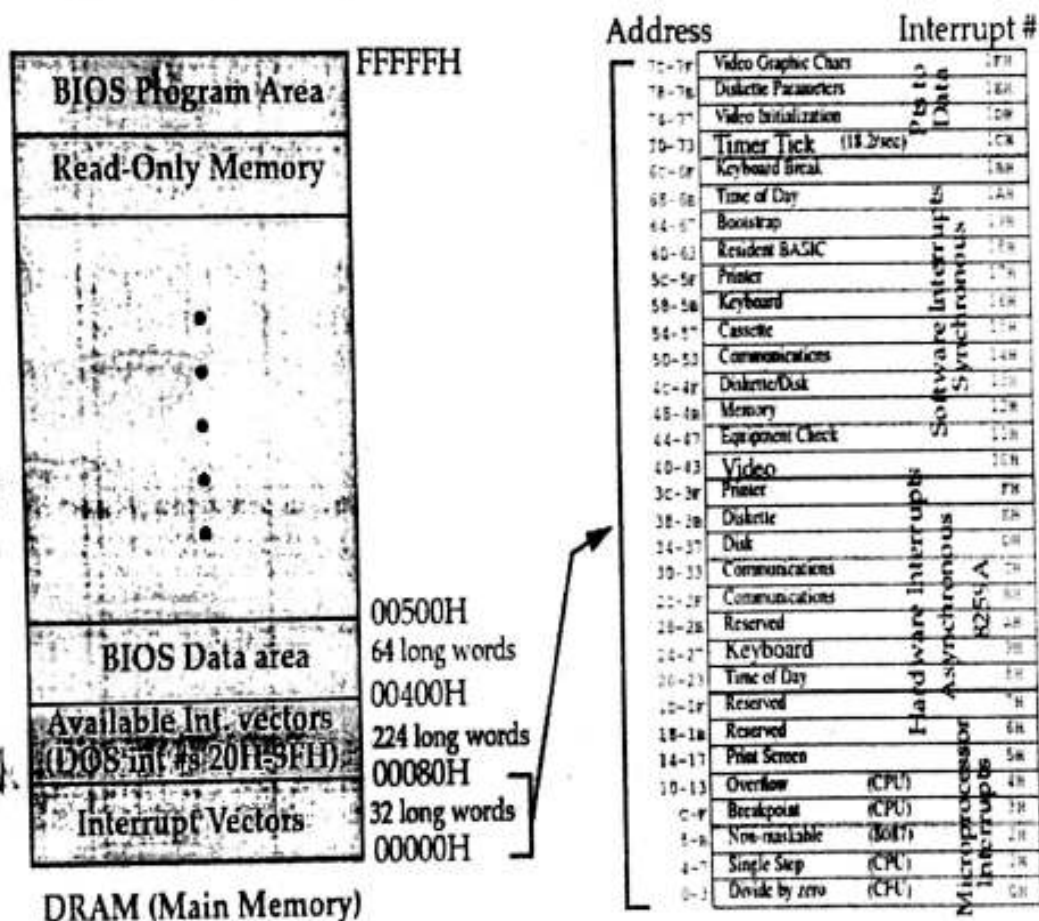
1. The contents of the FLAG REGISTERS are pushed onto the stack
2. Both the Interrupt (IF) and (TF) flags are cleared. This disables the INTR pin and the trap or single-step feature. (Depending on the nature of the interrupt, a programmer can unmask the INTR pin by the STI instruction)
3. The contents of the code segment register (CS) is pushed onto the stack.
4. The contents of the instruction pointer (IP) is pushed onto the stack.
5. The interrupt vector contents are fetched, and then placed into both IP and CS so that the next instruction executes at the interrupt service procedure addressed by the interrupt vector.
6. While returning from the Interrupt-service routine by the instruction IRET, flags return to their state prior to the interrupt and operation restarts at the prior IP address.

Discuss the two interrupts HLT and WAIT?

On execution of **HLT** (halt) instruction by 8086, CPU suspends its instruction execution and enters into an idle state. It waits for either an external hardware interrupt or a reset input (interrupt). When any one of these occurs, CPU starts executing again.

When the **WAIT** instruction is executed by 8086, it internally checks the logic level existing at its TEST input. If TEST is at logic 1 state, then CPU goes into an idle state. When TEST input assumes a zero state, execution resumes from the next sequential instruction in the program.

Interrupt Vectors (DOS PC)



Interrupt Vector

Memory Address	Table Entry	Vector Definition	
3F	CS 32	Vector 32 _u	User Available
3E	IP 32		
3D	CS 31	Vector 31 _u	
3C	IP 31		
18	CS 5	Vector 5	Reserved
14	IP 5		
12	CS 4	Vector 4 - Overflow	
10	IP 4		
0E	CS 3	Vector 3 - Breakpoint	
0C	IP 3		
0A	CS 2	Vector 2 - NMI	
08	IP 2		
06	CS 1	Vector 1 - Single Step	
04	IP 1		
02	CS Value - Vector 0 (CS 0)		Vector 0 - Divide Error
00	IP Value - Vector 0 (IP 0)		

2 Bytes

INT 00 (divide error)

; WRITE A DIVIDE ERROR ISR

Prompt db 'Division by zero attempted\$'

Divert. PUSH DX

Mov ah,00h

```
Mov dx, offset prompt
```

int 21h

POP DX

PRINT 01 (Single Step)

*In executing a sequence of instructions, there is often a need to examine the contents of the CPU's registers and system memory.

* This is done by executing one instruction at a time and then inspecting the registers and memory

* This is called the tracing or the single stepping

*TF must be set (DS of the flag register)

1 PUSHF

POP AN

OR AX,0000000100000000B

PUSH AX

POPF

- BIOS and DOS programming in Assembly
- BIOS INT 10H
- DOS INT 21H

BIOS AND DOS PROGRAMMING IN ASSEMBLY

- BIOS and DOS contain some very useful subroutines, which can be used through INT (interrupt) instruction.
- The INT instruction works like a FAR call. When it is invoked, it saves CS:IP and the flags on the stack and goes to the subroutine associated with the interrupt.

INT xx .the interrupt number can be 00 – FFH (256 possible interrupts)

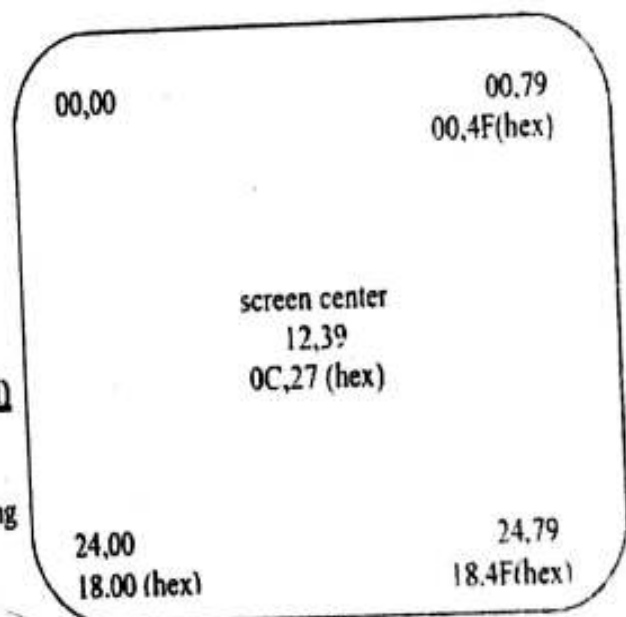
BIOS INT 10H PROGRAMMING

- INT 10H subroutines are in the ROM BIOS of the 80x86-based IBM PC.
- Depending on the value put in AH many function associated with the manipulation of screen text or graphics is performed.
- Among these functions, clearing the screen, changing the cursor position, change the screen color and drawing lines on the screen.

Monitor screen in text mode

- In normal text mode the screen is divided into 80 columns and 25 rows.
- Top left = 00,00
Bottom left = 24,00 (decimal)
Bottom right = 24,79 (decimal)
- Clearing the screen (INT 10H function 06H)
- AH=06 Scroll window up
- To clear the screen with INT 10H the following registers must contain certain values.

AH=06. AL=00. BH=07. CX=0000
DH=24. DL=79



The code:

```
MOV AH,06 ;AH=06 select the scroll function
MOV AL,00 ;number of lines to scroll (if AL=00 the entire page)
MOV BH,07 ;the display attribute (BH=07 normal)
MOV CH,00 ;row value of the start point
MOV CL,00 ;column value of the start point
MOV DH,24 ;row value of the ending point
MOV DL,79 ;column value of the ending point
INT 10H ;invoke the interrupt
```

More efficient coding:

```
MOV AX,0600H ;scroll entire screen
MOV BH,07 ;normal attribute
MOV CX,0000 ;start at 00,00
MOV DX,184FH ;end at 24,79 (hex=18,4F)
INT 10H ;invoke the interrupt
```

- **INT 10H function 02: setting the cursor to a specific location**

AH=02 Set cursor position

BH = page number (BH=00) ; 00 represents the current viewed page.

DH = row

DL = column

Ex: Write the code to set the cursor position to row = 15 (= 0FH) and column = 25 (= 19H)

```
MOV AH,02      ;set cursor option
MOV BH,00      ;page 0
MOV DH,15      ;row position
MOV DL,25      ;column position
INT 10H        ;invoke interrupt 10H
```

Ex: Write a program segment to (1) clear the screen and (2) set the cursor at the center of the screen.

;clearing the screen

```
MOV AX,0600H   ;scroll the entire page
MOV BH,07      ;normal attribute
MOV CX,0000H   ;row and column of the top left
MOV DX,184FH   ;row and column of the bottom right
INT 10H        ;invoke interrupt 10H
```

;setting the cursor to the center of the screen

```
MOV AH,02      ;set cursor option
MOV BH,00      ;page 0
MOV DH,12      ;center row position
MOV DL,39      ;center column position
INT 10H        ;invoke interrupt 10H
```

- **INT 10H function 03: get current cursor position**

AH=03 Read cursor position and size

Ex:

```
MOV AH,03      ;option 03 of BIOS INT 10H (read cursor position and size)
MOV BH,00      ;choose current (00) page
INT 10H        ;interrupt 10H routine
```

After the execution of the above program: DH = current row, DL = current column CX will provide info about the shape of the cursor

DOS INT 21H PROGRAMMING

➤ INT 21H subroutines are provided by DOS Operating system.

- Depending on the value put in AH many functions such as inputting data from the keyboard and displaying it on the screen can be performed.

INT 21H option 09: outputting a string of data to the monitor

➤ INT 21H can be used to send a set of ASCII data to the monitor.

➤ Register settings before INT 21H is invoked:

AH=09

DX = the offset address of the ASCII data to be displayed.

- INT 21H option 09 will display the ASCII data string pointed at by DX until it encounters the dollar sign '\$'. Note that this option cannot display '\$' character on the screen.

Ex:
 DB 'I love MICROPROCESSORS','\$'

 MOV AH,09 ;option 09 to display string of data
 MOV DX,OFFSET DATA_ASC ;DX offset address of data
 INT21H ;invoke the interrupt

INT 21H option 02: outputting a single character to the monitor

- To do that: AH=02 (AH is given 02)
 DL - is loaded with the ASCII character to be displayed.
 INT 21H is invoked.

03 string out
 02 single out
 07 keyboard
 01 ASCII

Ex:
 MOV AH,02 ;option 02 displays one character
 MOV DL,'Y' ;DL holds the character to be displayed
 INT 21H ;invoke the interrupt.

int 21h

- * This option can be used to display '\$' sign on the monitor.

INT 21H option 01: Keyboard input with echo (inputting a single character with echo)

- This function waits until a character is input from the keyboard, then echoes (displays) it to the monitor.
- After the interrupt the character will be in AL.

Ex:
 MOV AH,01 ;option 01 inputs one character
 INT 21H ;after the interrupt, AL = input character (ASCII)

INT 21H option 07: Keyboard input without echo

- This function waits until a character is input from the keyboard, then character is not displayed (echoed) to the monitor.
- After the interrupt the character will be in AL.

Ex:
 MOV AH,07 ;keyboard input without echo
 INT 21H ;after the interrupt, AL = input character (ASCII)

04 1-23

01
 07
 02
 09

org 100h

```
MOV AH, 0Eh ; select sub-function.  
INT 10h / 0Eh sub-function  
receives an ASCII code of the  
character that will be printed
```

```
in AL register.  
MOV AL, 'H' ; ASCII code: 72  
INT 10h ; print it!
```

```
MOV AL, 'e' ; ASCII code: 101  
INT 10h ; print it!
```

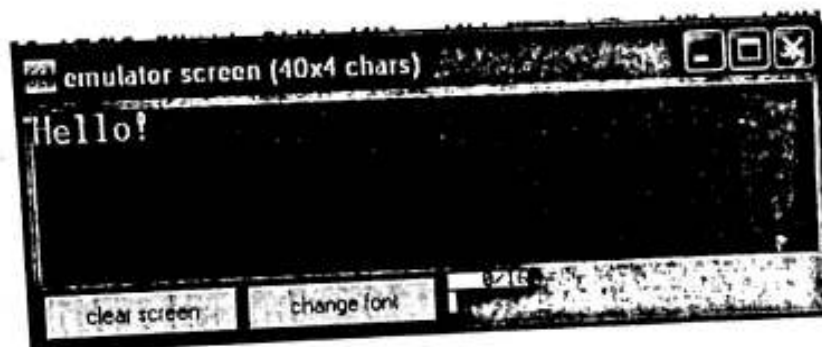
```
MOV AL, 'l' ; ASCII code: 108  
INT 10h ; print it!
```

```
MOV AL, 'l' ; ASCII code: 108  
INT 10h ; print it!
```

```
MOV AL, 'o' ; ASCII code: 111  
INT 10h ; print it!
```

```
MOV AL, '!' ; ASCII code: 33  
INT 10h ; print it!
```

```
RET ; returns to operating system.
```



0013
.
MODEL SMALL

; Display the string : Hello!

.DATA

msg db 'Hello!','\$'

.CODE

.startup

mov ax,@data

mov ds,ax

lea dx,msg ;or mov dx,offset msg

mov ah,9

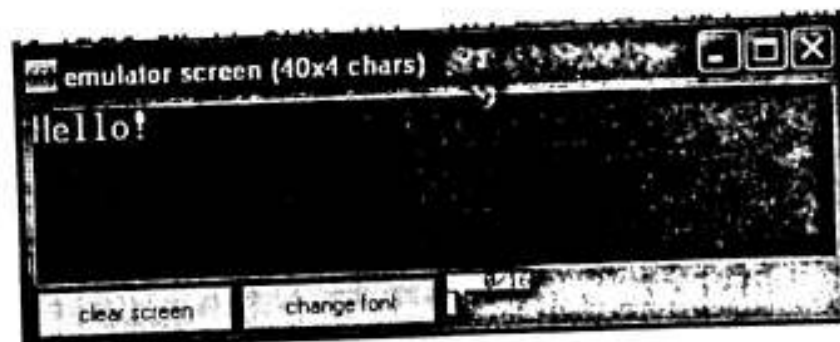
int 21h

; The following 2 statements return control
; back to DOS since we are finished

mov ax,4c00h

int 21h

end




```

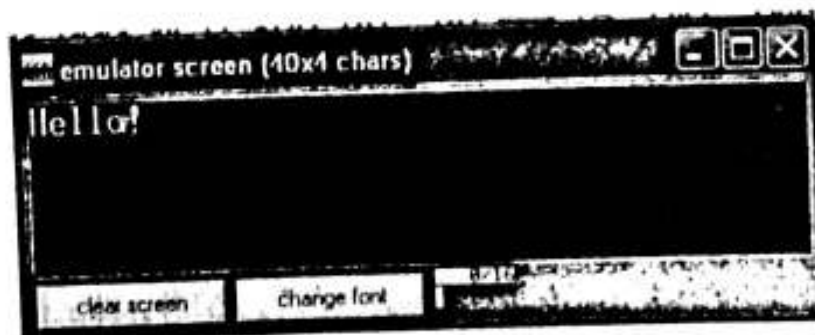
; "Hello!" without using INT 21h
name "hello"
org 100h ; compiler directive to make tiny.com file.
; execution starts here, jump over the data string:
jmp start

; data string:
msg db 'Hello!', 0dh, 0ah, 0
; 0dh, 0ah - is the code
;         for standard new
;         line characters:
;         0dh - carriage return.
;         0ah - new line.

start:
; set the index register:
mov     si, 0
next_char:
; get current character:
mov     al, msg[si]
; is it zero?
; if so stop printing:
cmp     al, 0
je      stop

; print character in teletype mode:
mov     ah, 0eh
int     10h
; update index register by 1:
inc     si
; go back to print another char:
jmp     next_char
stop:   mov     ah, 0 ; wait for any key press.
        int     16h
; exit here and return control to operating system...
ret
end ; to stop compiler.

```



; a tiny example of multi segment executable file.
; data is stored in a separate segment, segment registers must be set correctly.

name "testexe"

data segment
msg db "hello!", '\$'
ends

stack segment
db 30 dup(0)
ends

code segment
start:

; set segment registers:
mov ax, data
mov ds, ax
mov es, ax

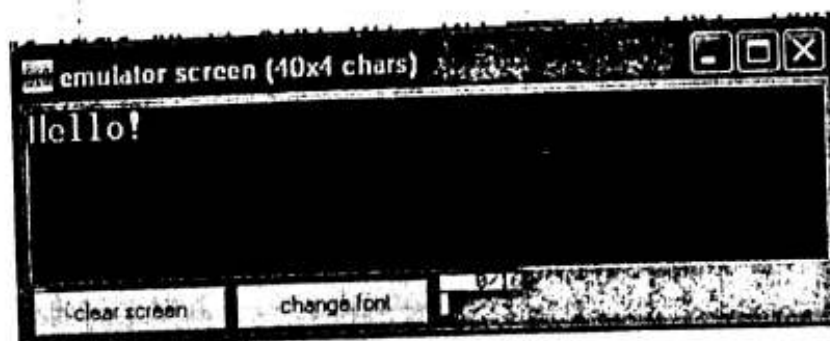
; print "hello!":
lea dx, msg
mov ah, 09h
int 21h

; wait for any key...
mov ah, 0
int 16h

; return control to os:
mov ah, 4ch
int 21h

ends

end start ; set entry point and stop the assembler.



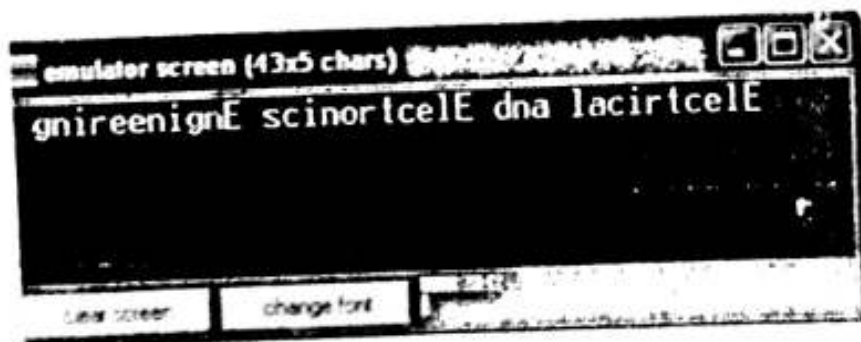
; Write a program to reverse the string

```
.model small
.stack 1000h
.data
msg db 'Electrical and Electronics Engineering'
length equ $-msg
.code
mov ax,@data
mov ds,ax
```

```
mov dx,0
mov si, offset msg
mov cx, length
```

```
add si,cx
back: mov dl,[si]
mov ah,02h
int 21h
dec si
loop back
mov ax,4c00h
int 21h
```

```
ret
```



1
; Print char. by char for any string
name "charchar"

org 100h
print_new_line macro

mov dl, 13
mov ah, 2
int 21h
mov dl, 10
mov ah, 2
int 21h

endm

mov dx, offset msg1
mov ah, 9
int 21h

; input the string:

mov dx, offset s1
mov ah, 0ah
int 21h

; get actual string size:

xor cx, cx

mov cl, s1[1]

print_new_line

mov bx, offset s1[2]

print_char:

mov dl, [bx]

mov ah, 2

int 21h

print_new_line

inc bx

loop print_char

; wait for any key...

mov ax, 0

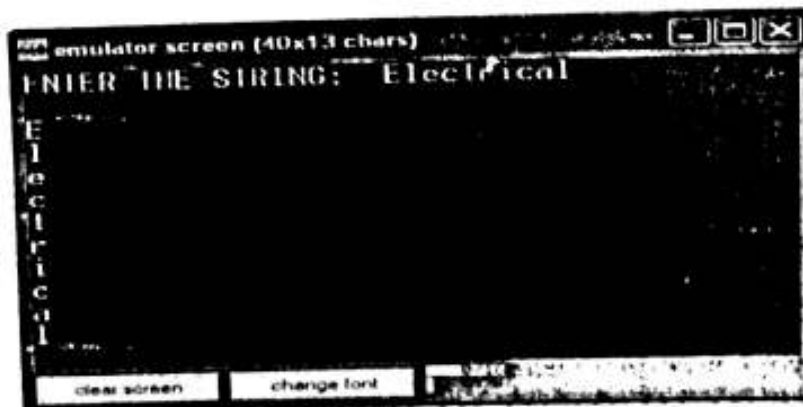
int 16h

ret

msg1 db "ENTER THE STRING: \$"

s1 db 100 dup(' ') ; Space

end



; this sample shows how to use scasd instruction to find a symbol.
org 100h

; set forward direction:
cld

; set counter to string size:
mov cx, 10

; load string address into es:di
mov ax, cs
mov es, ax

lea di, str1
; we will look for the character in string:
mov al, find_what

repne scasd
jz found

not_found:

; "no" - not found!
mov dx, offset s_not
mov ah, 9
int 21h

jmp exit_here

found:

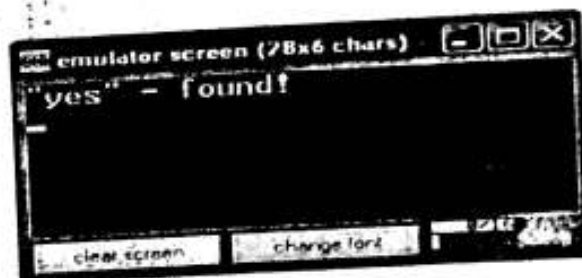
; "yes" - found!
mov dx, offset s_found
mov ah, 9
int 21h

; di contains the address of searched character:
dec di

; wait for any key press...
mov ah, 0
int 16h

exit_here:
ret

str1 db 'aaabbbxddd'
s_found db '"yes" - found!', 0Dh, 0Ah, '\$'
s_not db '"no" - not found!', 0Dh, 0Ah, '\$'
find_what equ 'x'



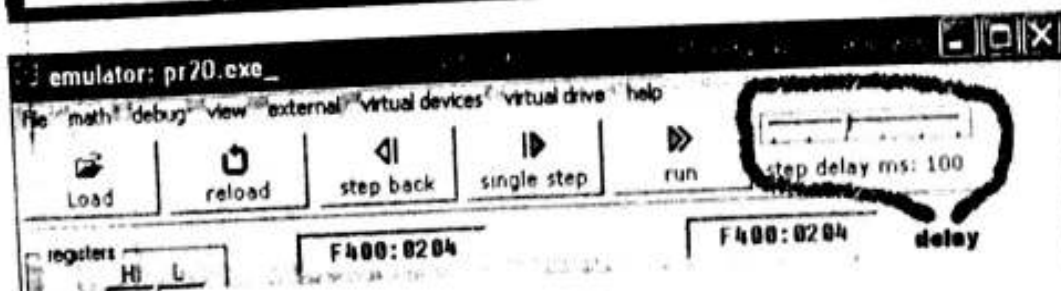
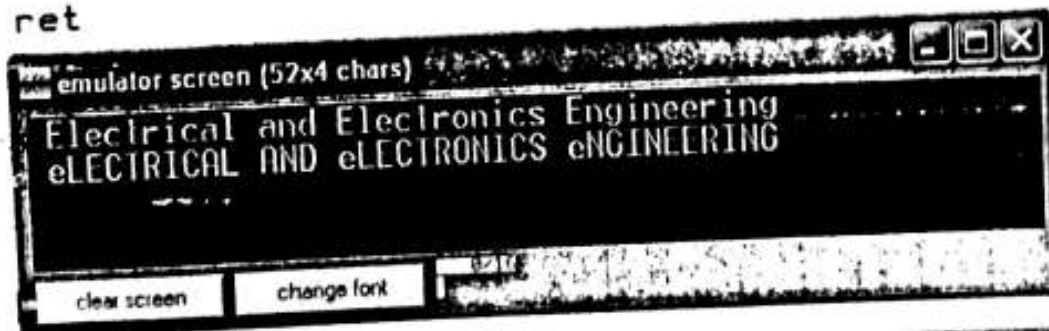
Write a program to convert upper two lower and lower to upper
delay time = 100 ms

```
.model small
stack 1000h
.data
msg db 'Electrical and Electronics Engineering'
    db 0dh,0ah
length equ $-msg
.code
mov ax,@data
mov ds,ax
mov dx,0
mov si,offset msg
mov cx,length

back: mov dl,[si]
     mov ah,02h
     int 21h
     inc si
     loop back

mov si,offset msg
mov cx,length
back1: mov dl,[si]
     xor dl,00100000b
     mov ah,02h
     int 21h
     inc si
     loop back1
mov ax,4c00h
int 21h

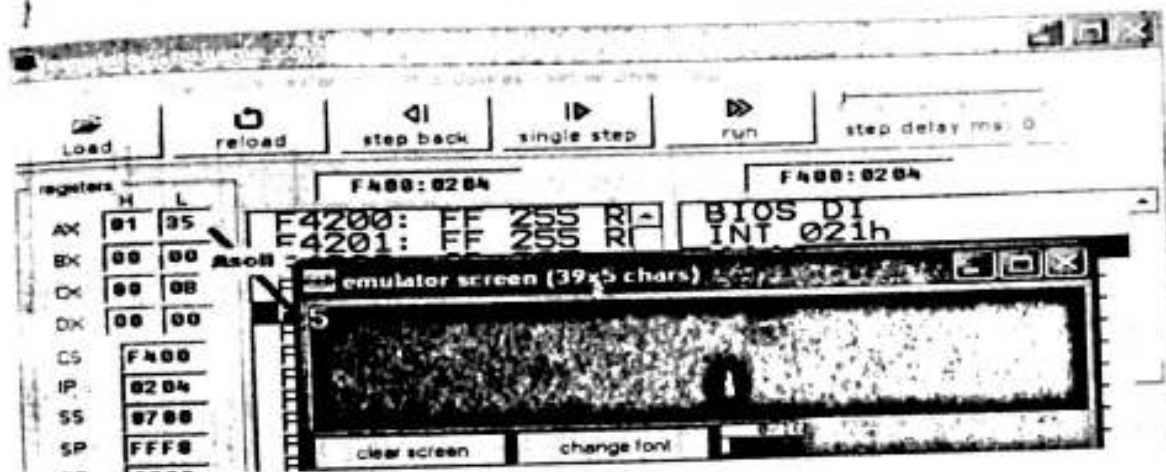
ret
```




```

; To read a character from keyboard
; you can use the DOS function call #1
; Ascii code of the key pressed store in AL
mov ah, 01h ; keyboard input with echo
int 21h      ; ah=07 keyboard input without echo

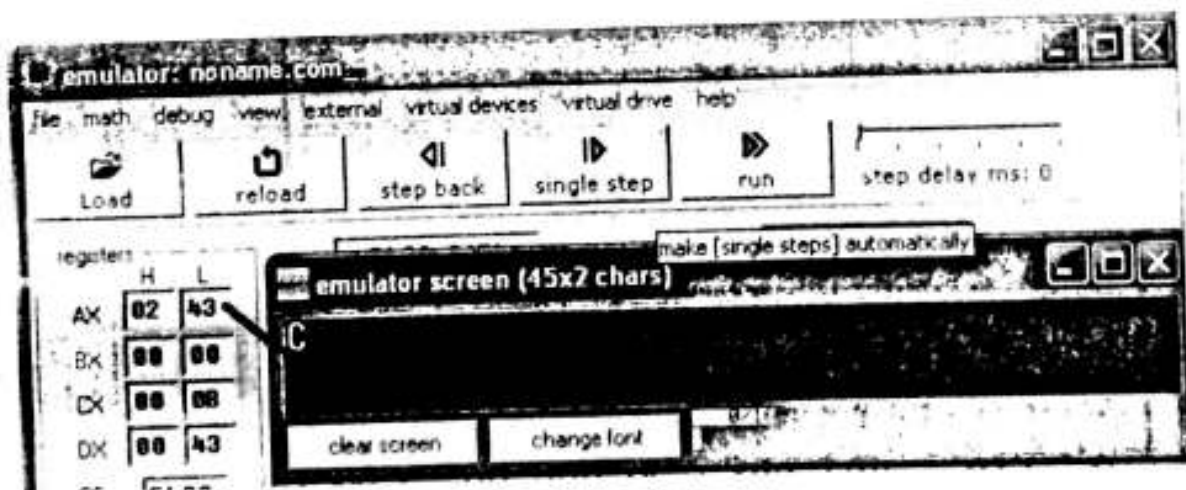
```



```

; To write a character to the screen
; you can use the DOS function call #2
mov dl, 'C'
mov AH, 2
int 21h

```



```

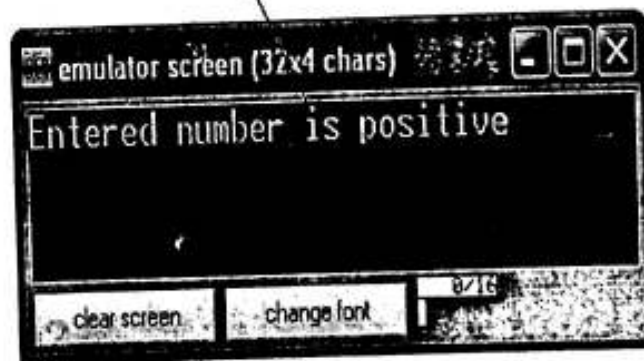
.model small
.data
msg1 db 'Entered number is positive $'
msg2 db 'Entered number is negative $'
input db 15
; .stack
.code
mov ax, @data
mov ds, ax
mov al, input
rol al, 01h
jc next

lea dx, msg1
mov ah, 09h
int 21h
jmp last

next: lea dx, msg2
mov ah, 09h
int 21h

last: mov ah, 4ch
int 21h

```



```

.model small
.data
msg1 db 'Entered number is positive $'
msg2 db 'Entered number is negative $'
input db -15

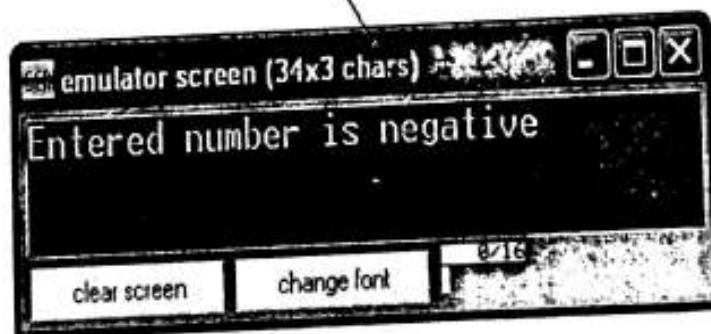
; .stack
.code
mov ax, @data
mov ds, ax
mov al, input
rol al, 01h
jc next

lea dx, msg1
mov ah, 09h
int 21h
jmp last

next: lea dx, msg2
mov ah, 09h
int 21h

last: mov ah, 4ch
int 21h

```



Pr 22

```

; Write ALP to read two decimal numbers
; one digit each from keyboard and print
; the sum of two number on screen

```

```

.model small
.data
x db ?
y db ?

.code
mov ax,@data
mov ds,ax
mov ah,1
int 21h          ; read first number
sub al,30h       ; convert to decimal
mov x,al

call return      ; New line

mov ah,1
int 21h          ; read second number
sub al,30h       ; convert to decimal
mov y,al

call return

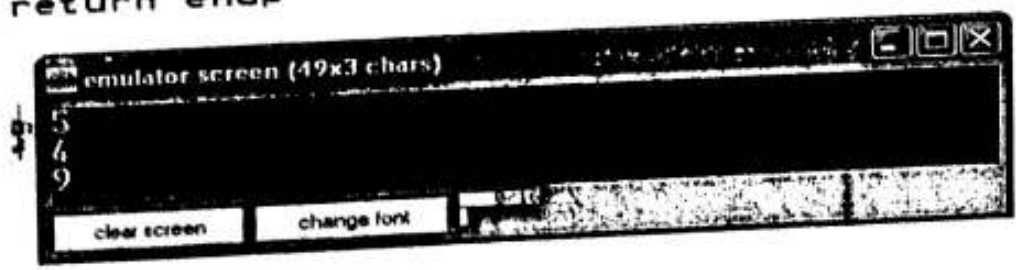
mov al,y
add al,x
mov dl,al
add dl,30h       ; convert sum to ASCII

mov ah,2
int 21h

mov ah,4ch
int 21h

return proc
    mov dl,0ah
    mov ah,2
    int 21h
    mov dl,0dh
    mov ah,2
    int 21h
    ret
return endp

```



Write ALP to read two decimal numbers
 one digit each from the keyboard
 and print the sum of the two numbers. All done.

```
.model small
.stack 100h
.data
x db ?
y db ?
z db ?
msg1 db 'please enter the first number #'
msg2 db 'please enter the second number #'
msg3 db 'the sum is #'

.code
mov ax,@data
mov ds,ax      ; initialized ds
lea dx,msg1
call PMSG      ; print: please enter the first number
call return    ; Go to next line on screen
call RD10      ; Read first number
mov x,al

    call return

lea dx,msg2
call PMSG      ; print: please enter the second number
call return    ; Go to next line on screen
call RD10      ; Read second number
mov y,al
add al,x

mov z,al
call return

mov dx, offset msg3
call PMSG      ; print: the sum is
mov dl,z
call P10
mov ah,4ch
int 21h

PMSG proc
    mov ah,9
    int 21h
    ret
PMSG endp
```

```

RD1D proc
    mov ah,1
    int 21h
    sub al,30h
    ret

```

```
RD1D endp
```

```

return proc
    mov dl,0ah
    mov ah,2
    int 21h
    mov dl,0dh
    mov ah,2
    int 21h
    ret

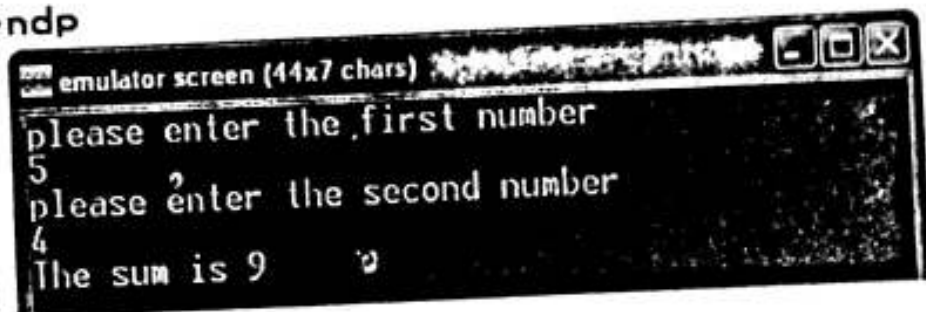
```

```
return endp
```

```

PTD proc
    add dl,30h
    mov ah,2
    int 21h
    ret
PTD endp

```



```

emulator screen (44x7 chars)
please enter the first number
5
please enter the second number
4
The sum is 9

```



```

.model small
.data
msg db 'please enter any number,$'
msg1 db 'entered number is odd $,'
msg2 db 'entered number is even $,'
input db 7

.code
mov ax,@data
mov ds,ax

lea dx,msg
call PMSG ; print:please enter any number
call return ; Go to next line on screen

call RD1D ; Read number
mov input,al
call return
mov al,input
ror al,01h
jnc next

lea dx,msg1
call PMSG

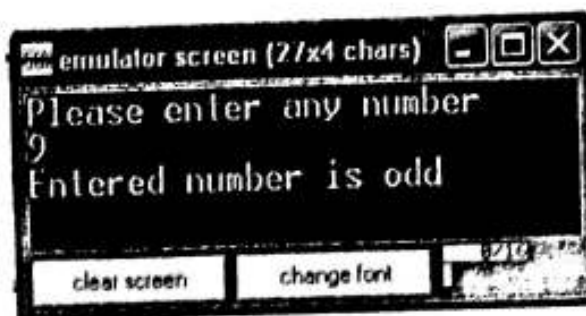
jmp last
next: lea dx,msg2
call PMSG
last: mov ah,4ch
int 21h

PMSG proc
mov ah,9
int 21h
ret
PMSG endp

return proc
mov dl,0ah
mov ah,2
int 21h
mov dl,0dh
mov ah,2
int 21h
ret
return endp

RD1D proc
mov ah,1
int 21h
sub al,30h
ret
RD1D endp

```



8086 /8088 Hardware Specification

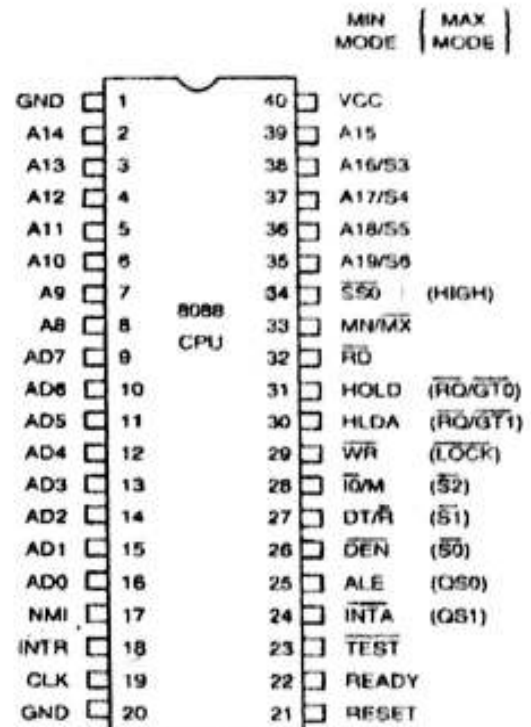
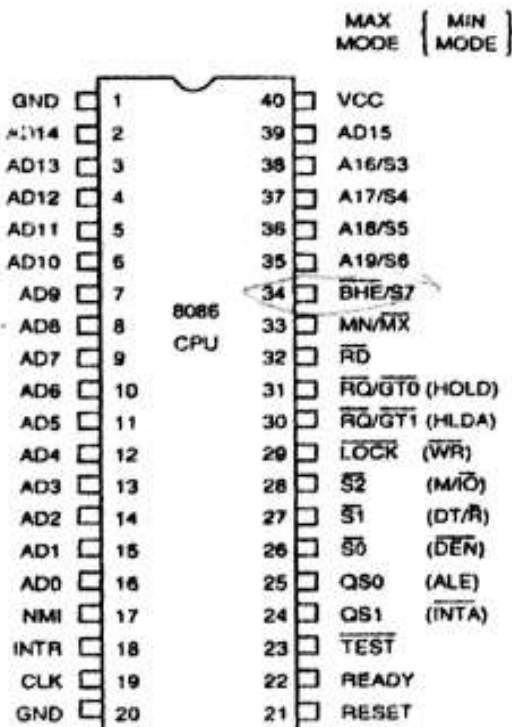
In this lecture, we cover the 8086 microcomputer from the hardware point of view. The 8086, announced in 1978, was the first 16-bit microprocessor introduced by Intel Corporation. The 8086 is manufactured using *high-performance metal-oxide semiconductor (HMOS) technology*, and the circuitry on its chips is equivalent to approximately 29000 transistors. It is housed in a 40-pin dual in-line package.

Pin outs and the pin functions

Figure below illustrates the pin-outs of the 8086 and 8088 microprocessors. As a close comparison reveals, there is virtually no difference between these two microprocessors—both are packaged in 40-pin dual in-line packages (DIPs).

the 8086 is a 16-bit microprocessor with a 16-bit data bus, and the 8088 is a 16-bit microprocessor with an 8-bit data bus. (As the pin-outs show, the 8086 has pin connections AD₀–AD₁₅, and the 8088 has pin connections AD₀–AD₇.) Data bus width is therefore the only major difference between these microprocessors.

There is, however, a minor difference in one of the control signals. The 8086 has an M/ $\overline{\text{IO}}$ pin, and the 8088 has an IO/ $\overline{\text{M}}$ pin. The only other hardware difference appears on Pin 34 of both chips: on the 8088, it is an $\overline{\text{SSO}}$ pin, while on the 8086, it is a $\overline{\text{BHE}}/\text{S}_7$ pin.



8086/88 Device Specifications

Both are packaged in DIP (Dual In-Line Packages).

- 8086: 16-bit microprocessor with a **16-bit** data bus
- 8088: 16-bit microprocessor with an **8-bit** data bus.

Both are 5V parts:

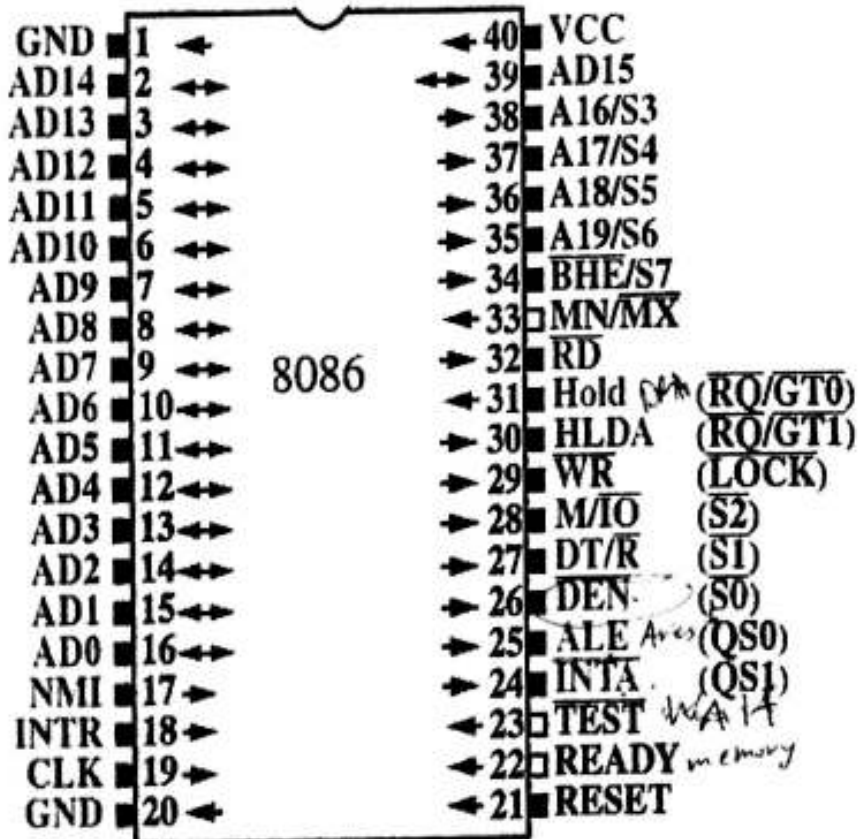
8086: Draws a maximum supply current of 360mA.

8088: Draws a maximum supply current of 340mA.

Input/Output levels:

INPUT		OUTPUT	
Logic level	Voltage	Logic level	Voltage
0	0.8V max	0	0.45V max
1	2.0V min	1	2.4V min

MIN MODE (MAX MODE)



8086/88 Pinout

Pin functions:

- AD15 AD0

Multiplexed address (ALE = 1) / data bus (ALE = 0).

- A19/S6-A16/S3 (multiplexed)

High order 4 bits of the 20-bit address OR status bits S6-S3.

- M/IO

Indicates if address is a Memory or IO address.

- RD

When 0, data bus is driven by memory or an I/O device.

- WR

Microprocessor is driving data bus to memory or an I/O device. When 0, data bus contains valid data.

- ALE (Address latch enable)

When 1, address data bus contains a memory or I/O address.

- DT/R (Data Transmit/Receive)

Data bus is transmitting/receiving data.

- DEN (Data bus Enable)

Activates external data bus buffers.

- S7, S6, S5, S4, S3, S2, S1, S0

(S7: Logic 1, S6: Logic 0.)

S5: Indicates condition of IF flag bits.

S4-S3: Indicate which segment is accessed during current bus cycle:

S4	S3	Function
0	0	Extra segment
0	1	Stack segment
1	0	Code or no segment
1	1	Data segment

S2, S1, S0: Indicate function of current bus cycle (decoded by 8288).

S2	S1	S0	Function	S2	S1	S0	Function
0	0	0	Interrupt Ack	1	0	0	Opcode Fetch
0	0	1	I/O Read ✓	1	0	1	Memory Read ✓
0	1	0	I/O Write ✓	1	1	0	Memory Write
0	1	1	Halt	1	1	1	Passive

Pin functions:

- INTR

When 1 and $IF=1$, microprocessor prepares to service interrupt. $INTA$ becomes active after current instruction completes.

- $INTA$

Interrupt Acknowledge generated by the microprocessor in response to INTR. Causes the interrupt vector to be put onto the data bus.

- NMI

Non-maskable interrupt. Similar to INTR except IF flag bit is not consulted and interrupt is vector 2.

- CLK

Clock input must have a duty cycle of 33% (high for $1/3$ and low for $2/3$ s)

- VCC/GND

Power supply (5V) and GND (0V).

- MN/\overline{MX}

Select minimum (5V) or maximum mode (0V) of operation.

- \overline{BHE}

Bus High Enable. Enables the most significant data bus bits ($D_{15}-D_8$) during a read or write operation.

- READY

Used to insert wait states (controlled by memory and IO for reads/writes) into the microprocessor.

- RESET

Microprocessor resets if this pin is held high for 4 clock periods. Instruction execution begins at $FFFF0H$ and IF flag is cleared.

- TEST

An input that is tested by the WAIT instruction. Commonly connected to the 8087 coprocessor.

- **HOLD**

Requests a direct memory access (DMA). When 1, microprocessor stops and places address, data and control bus in high-impedance state.

- **HLDA (Hold Acknowledge)**

Indicates that the microprocessor has entered the hold state.

- **$\overline{RQ}/\overline{GTI}$ and $\overline{RQ}/\overline{GT0}$**

Request/grant pins request/grant direct memory accesses (DMA) during maximum mode operation.

- **LOCK**

Lock output is used to lock peripherals off the system. Activated by using the LOCK: prefix on any instruction.

- **QS1 and QS0**

The queue status bits show status of internal instruction queue. Provided for access by the numeric coprocessor (8087).

QS_1	QS_0	Function
0	0	Queue is idle
0	1	First byte of opcode
1	0	Queue is empty
1	1	Subsequent byte of opcode

Bus cycle status (8088) using \overline{SSO}

$\overline{IO/\overline{M}}$	$\overline{DT/\overline{R}}$	\overline{SSO}	Function
0	0	0	Interrupt acknowledge
0	0	1	Memory read
0	1	0	Memory write
0	1	1	Halt
1	0	0	Opcode fetch
1	0	1	I/O read
1	1	0	I/O write
1	1	1	Passive

Note

- For example, we see that address bus lines A_0 through A_{15} and data bus lines D_0 through D_{15} are multiplexed. For this reason, these leads are labeled AD_0 through AD_{15} . By multiplexed we mean that the same physical pin carries an address bit at one time and the data bit at another time.

- The 8086 can be configured to work in either of two modes:

MIN and MAX Mode : Controlled through the $\overline{MN}/\overline{MX}$ pin.

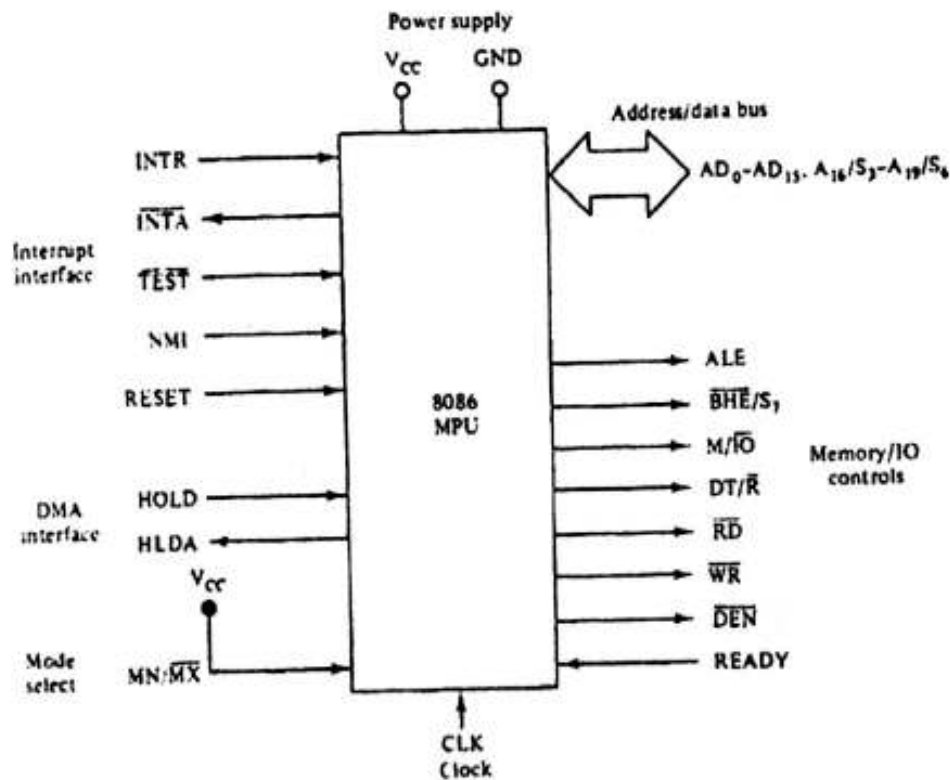
The **minimum mode** is selected by applying logic 1 to the $\overline{MN}/\overline{MX}$ input lead. Minimum mode 8086 systems are typically smaller and contain a single microprocessor. **Minimum mode** is cheaper since all control signals for memory and I/O are generated by the microprocessor.

The **maximum mode** is selected by applying logic 0 to the $\overline{MN}/\overline{MX}$ input lead. Maximum mode configures 8086 systems for use in larger systems and with multiple processors. **Maximum mode** is designed to be used when a coprocessor (8087) exists in the system.

Depending on the mode of operation selected, the assignments for a number of pins on the microprocessor package are changed.

Common signals		
Name	Function	Type
$AD_{15}-AD_0$	Address / data bus	Bidirectional, 3-state
$A_{19}/S_0-A_{16}/S_1$	Address / status	Output, 3-state
$\overline{MN}/\overline{MX}$	Minimum/Maximum mode control	Input
\overline{RD}	Read control	Output, 3-state
\overline{TEST}	Wait on test control	Input
\overline{READY}	Wait state control	Input
\overline{RESET}	System reset	Input
\overline{NMI}	Non-maskable interrupt request	Input
\overline{INTR}	Interrupt request	Input
\overline{CLK}	System clock	Input
V_{CC}	+5 volt	Input
\overline{GND}	Ground	Input

Minimum mode signals(MN/MX=V _{CC})		
Name	Function	Type
HOLD	Hold request	Input
HLDA	Hold acknowledgment	Output
WR	Write control	Output, 3-state
M/ \overline{IO}	IO/memory control	Output, 3-state
DT/ \overline{R}	Data transmit /receive	Output, 3-state
\overline{DEN}	Data enable	Output, 3-state
\overline{BHE} \ S7	Bank high enable/Status line 7	Output, 3-state
ALE	Address latch enable	Output
\overline{INTA}	Interrupt acknowledgment	Output



Minimum-Mode block diagrams

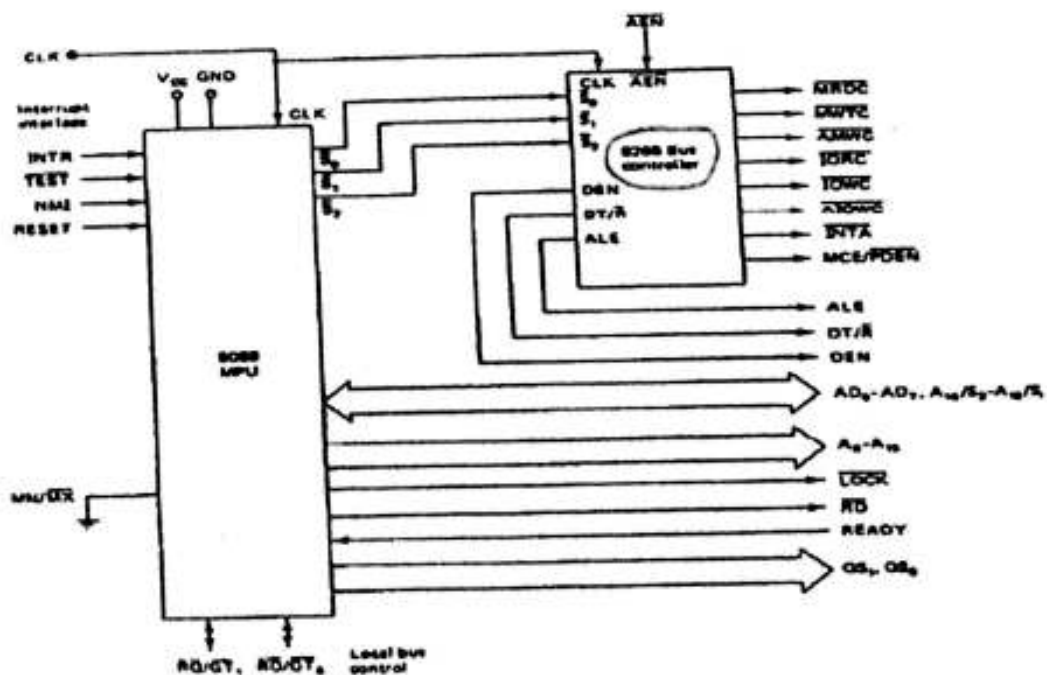
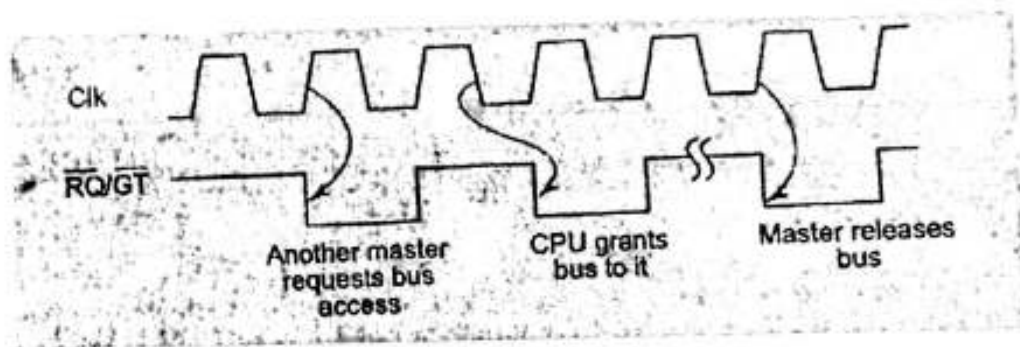
Some of the control signals must be generated externally, due to redefinition of certain control pins on the 8086.

The following pins are lost when the 8086 operates in **Maximum mode**.

- ALE
- WR
- IO/M
- DT/R
- DEN
- INTA

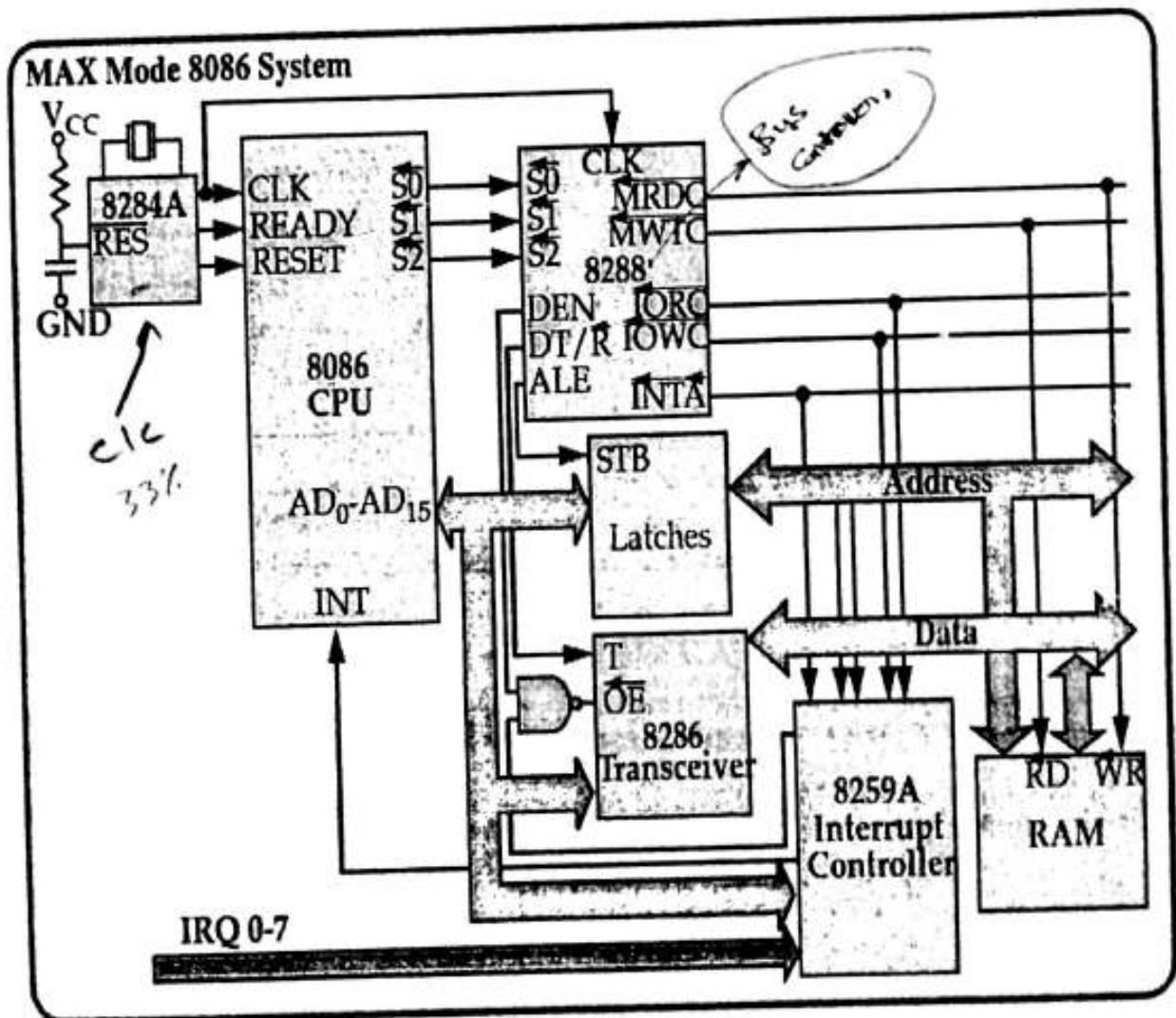
This requires an external bus controller: The **8288 Bus Controller**.

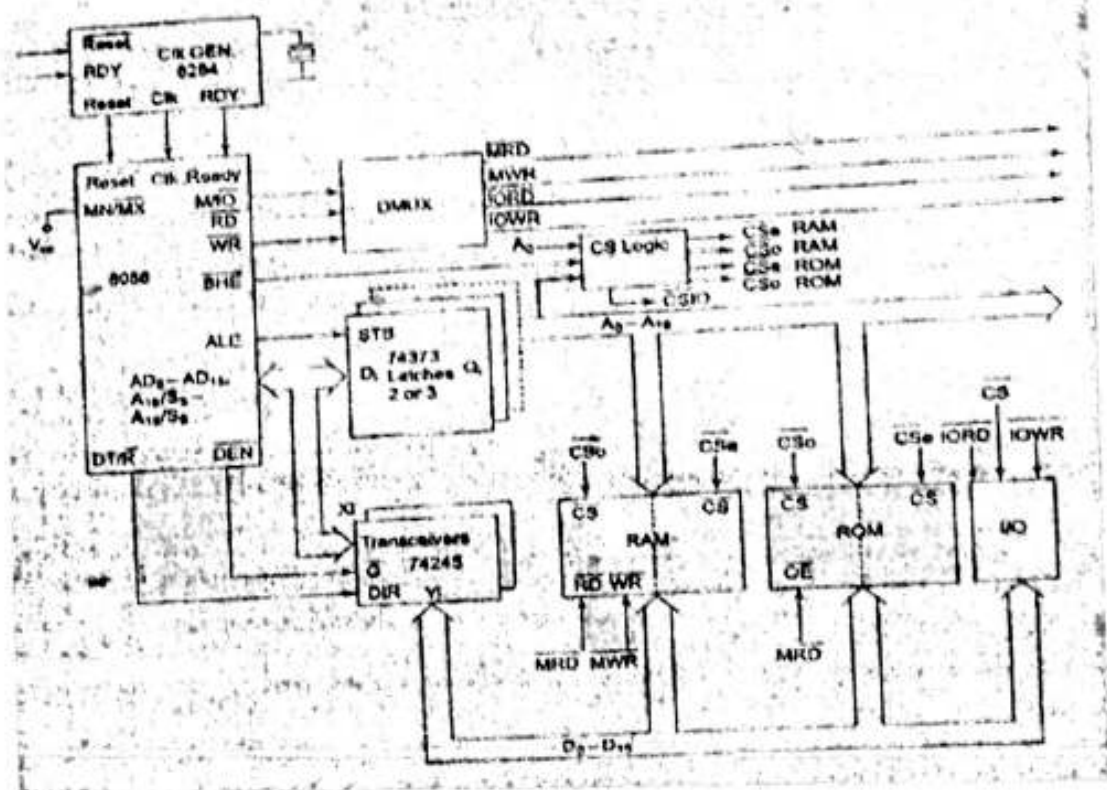
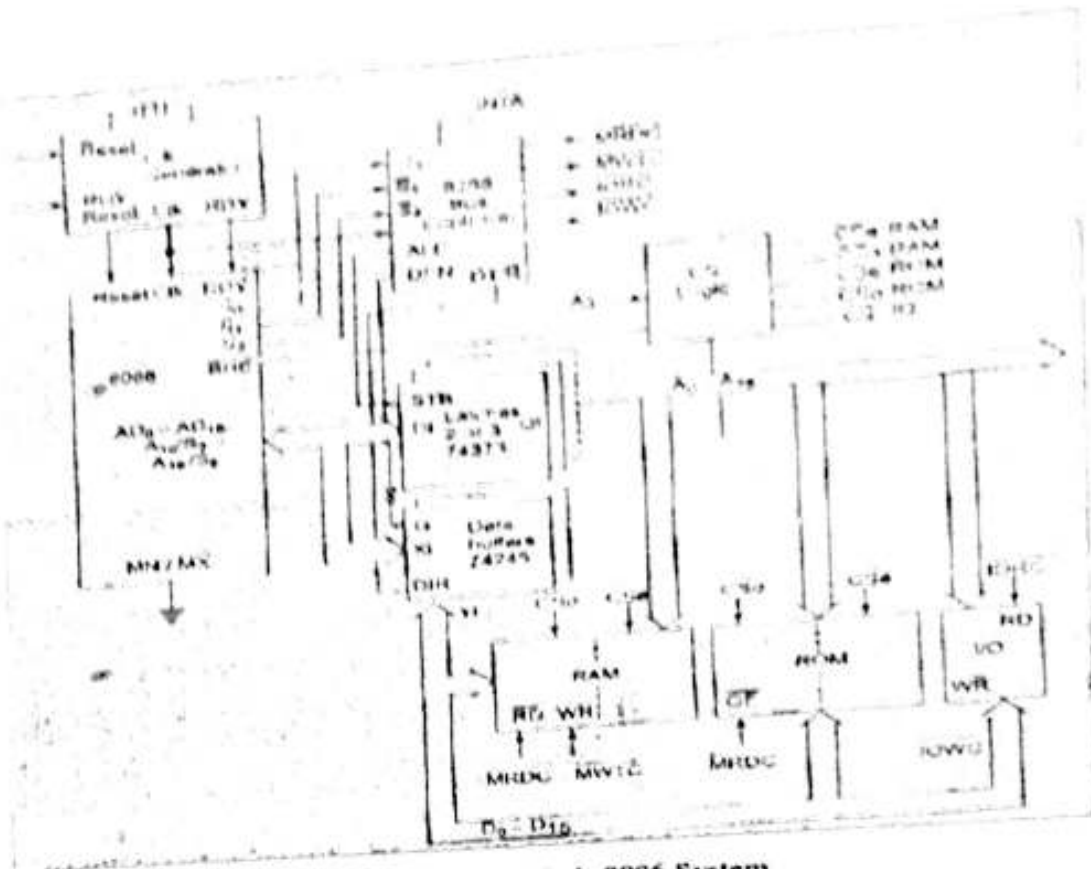
Maximum mode signals(MN/MX Ground)		
Name	Function	Type
RQ/GT1,0	Request/grant bus access control	Bidirectional
LOCK	Bus priority lock control	Output, 3-state
S ₂ - S ₀	Bus cycle status	Output, 3-state
QS1, QS0	Instruction queue status	Output



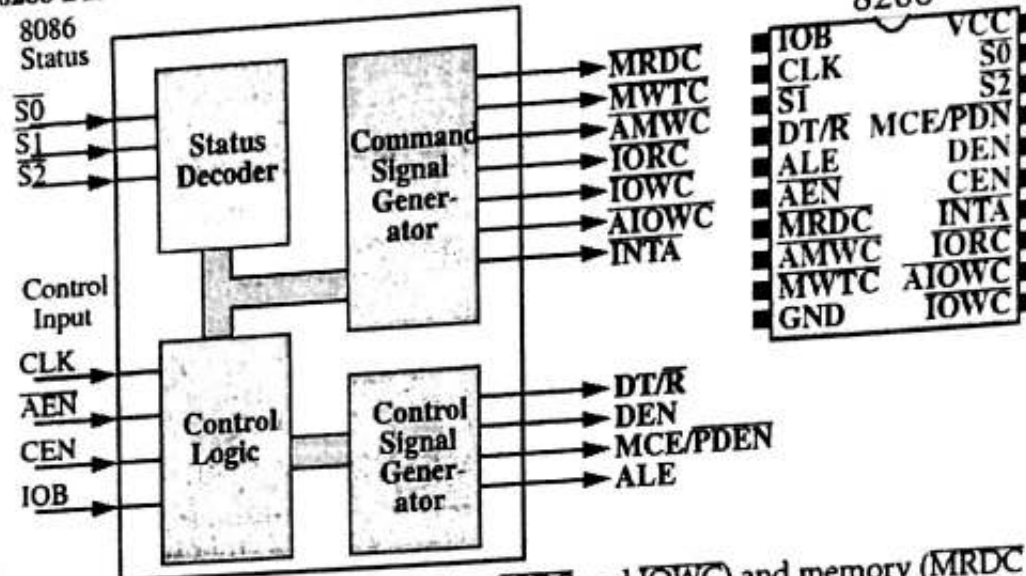
Status Inputs			CPU Cycle	8288 Command	Meaning
S2	S1	S0			
0	0	0	Interrupt Acknowledge	INTA	Interrupt acknowledge
0	0	1	Read I/O port	IORC	I/O read control
0	1	0	Write I/O port	IOWC, AIOWC	I/O write control, Advanced I/O write control
0	1	1	Halt	None	---
1	0	0	Instruction Fetch	MRDC	Memory read control
1	0	1	Read Memory	MRDC	Memory read control
1	1	0	Write Memory	MWTC, AMWC	Memory write control, advanced memory write control
1	1	1	Passive	None	---

Bus Status Codes





8288 Bus Controller



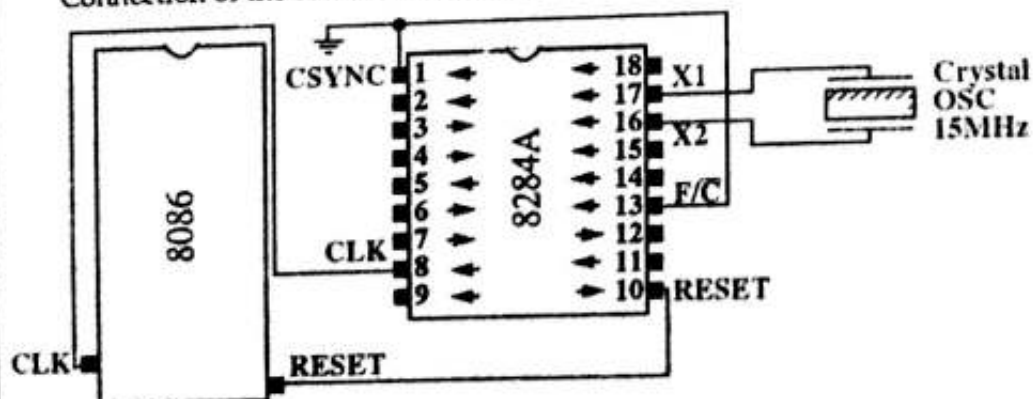
Separate signals are used for I/O (IORC and IOWC) and memory (MRDC and MWTC).
Also provided are advanced memory (AIOWC) and I/O (AIOWC) write strobes plus INTA.

8284A Clock Generator

Basic functions:

- Clock generation.
- RESET synchronization.
- READY synchronization.
- Peripheral clock signal.

Connection of the 8284 and the 8086.



BUS BUFFERING AND LATCHING

All computer systems: have three buses

- Address bus: provided memory and I/O with memory address or I/O port number.
- Data bus: transferred data between μ and memory or I/O.
- Control bus: provided control signal to memory and I/O.

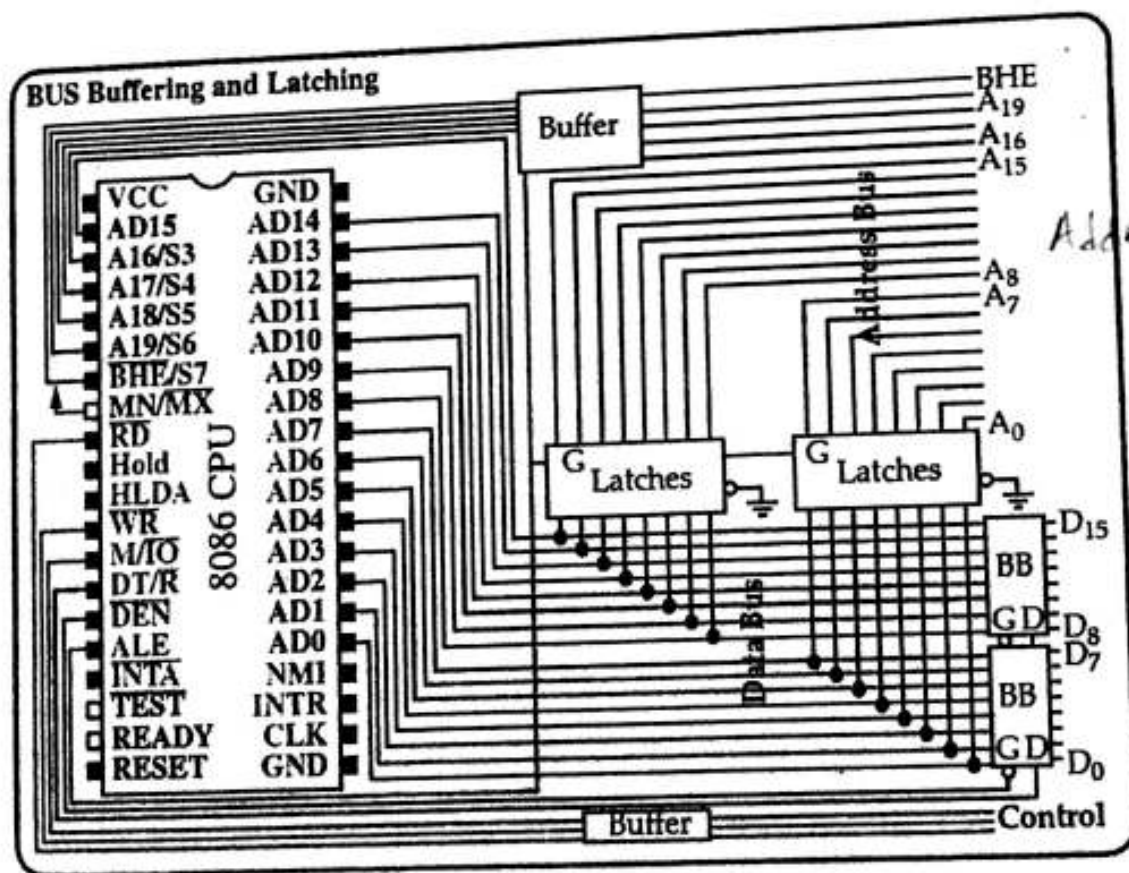
All signals MUST be buffered.

Latches buffer for A_0-A_{15} .

Control and $A_{16}-A_{19} + BHE$ are buffered separately.

Data bus buffers must be bi-directional buffers (BB).

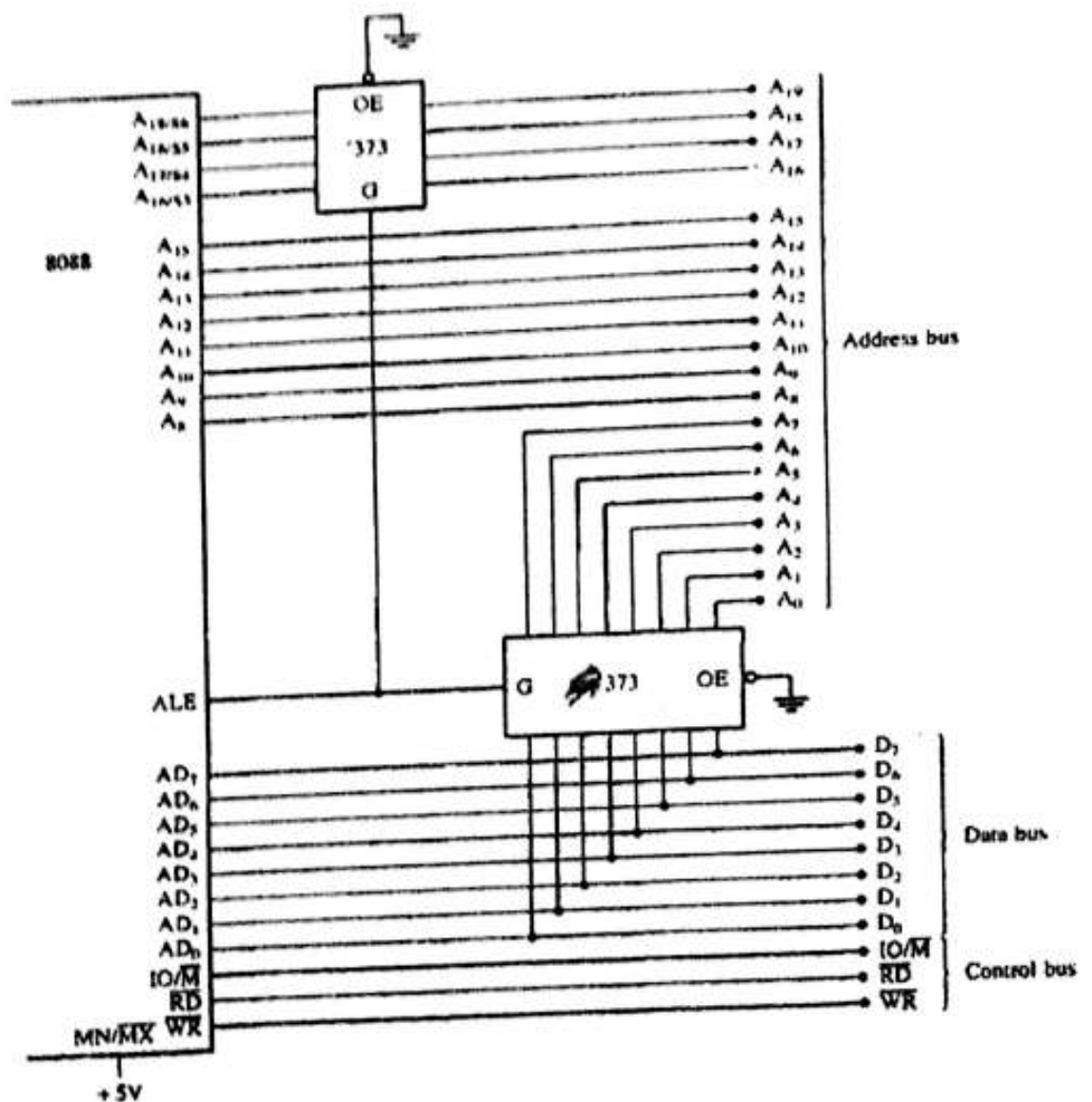
BHE : Selects the high-order memory bank.



Demultiplexing the 8088:

In this case, two 74LS373 transparent latches are used to de-multiplex the address/data bus connections AD_7-AD_0 and the multiplexed address/status connections $A_{19}/S_6-A_{16}/S_3$.

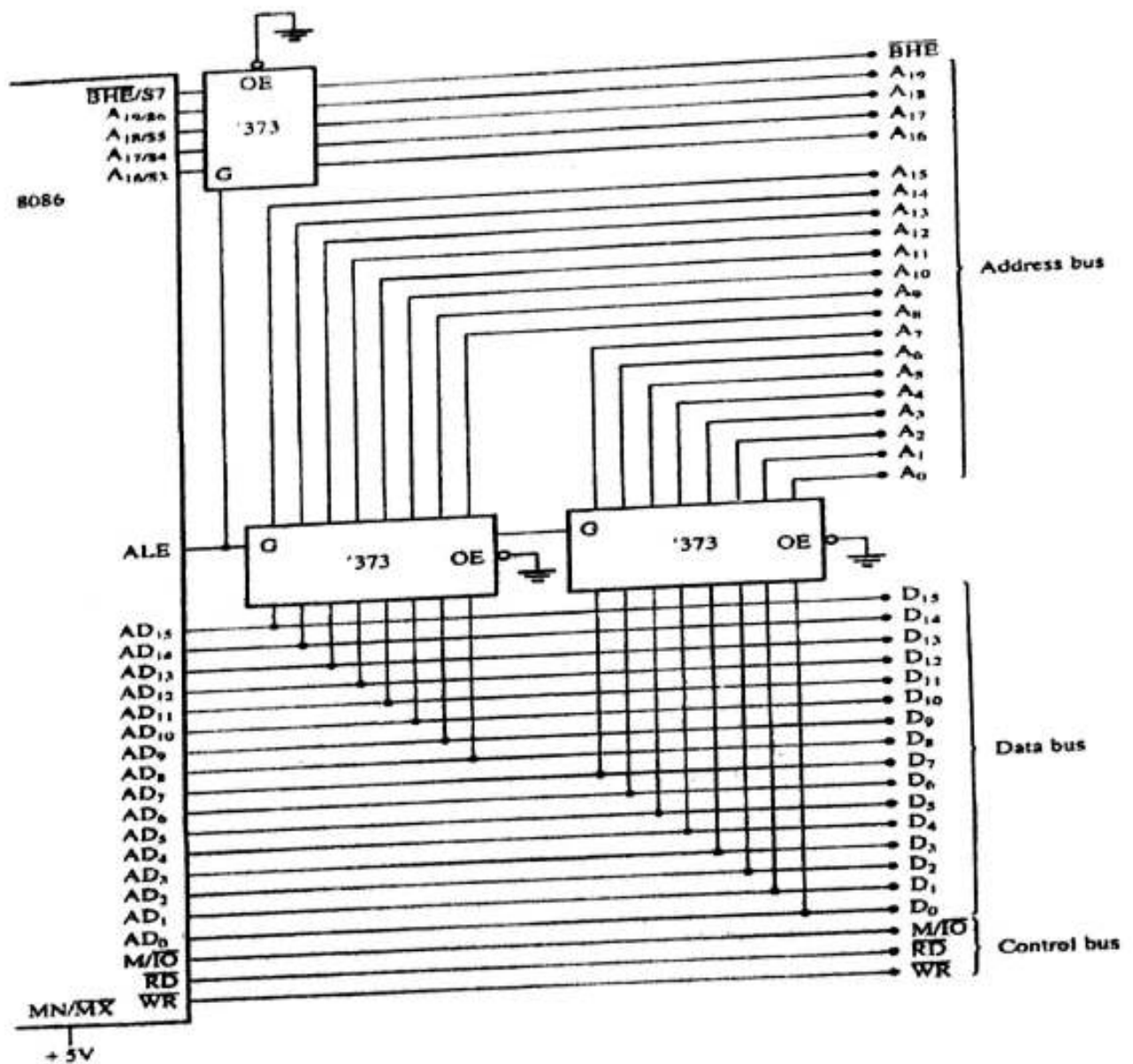
These transparent latches, which are like wires whenever the address latch enable pin (ALE) becomes a logic 1, pass the inputs to the outputs. After a short time, ALE returns to its logic 0 condition, which causes the latches to remember the inputs at the time of the change to a logic 0. In this case, A_7-A_0 are stored in the bottom latch and $A_{19}-A_{16}$ in the top latch.



Demultiplexing the 8086:

Demultiplexing the 8086. Like the 8088, the 8086 system requires separate address, data, and control buses. It differs primarily in the number of multiplexed pins. In the 8088, only AD_7-AD_0 and $A_{19}/S_6-A_{16}/S_3$ are multiplexed. In the 8086, the multiplexed pins include $AD_{15}-AD_0$, $A_{19}/S_6-A_{16}/S_3$, and \overline{BHE}/S_7 . All of these signals must be demultiplexed.

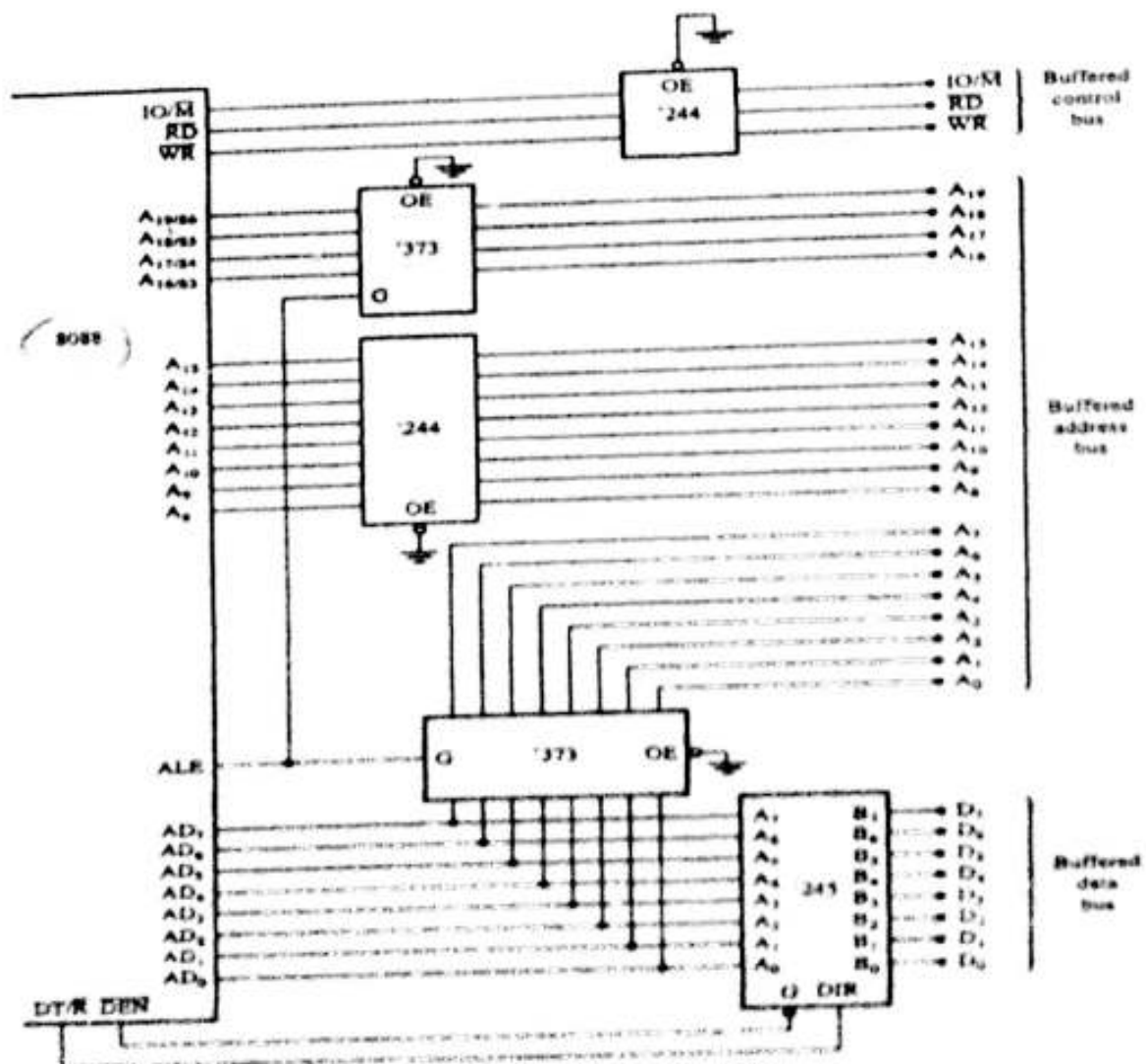
Figure illustrates a demultiplexed 8086 with all three buses: address ($A_{19}-A_0$ and \overline{BHE}), data ($D_{15}-D_0$), and control (M/\overline{IO} , \overline{RD} , and \overline{WR}).



The buffered System

A fully buffered signal will introduce a timing delay to the system. This causes no difficulty unless memory or I/O devices are used, which function at near the maximum speed of the bus.

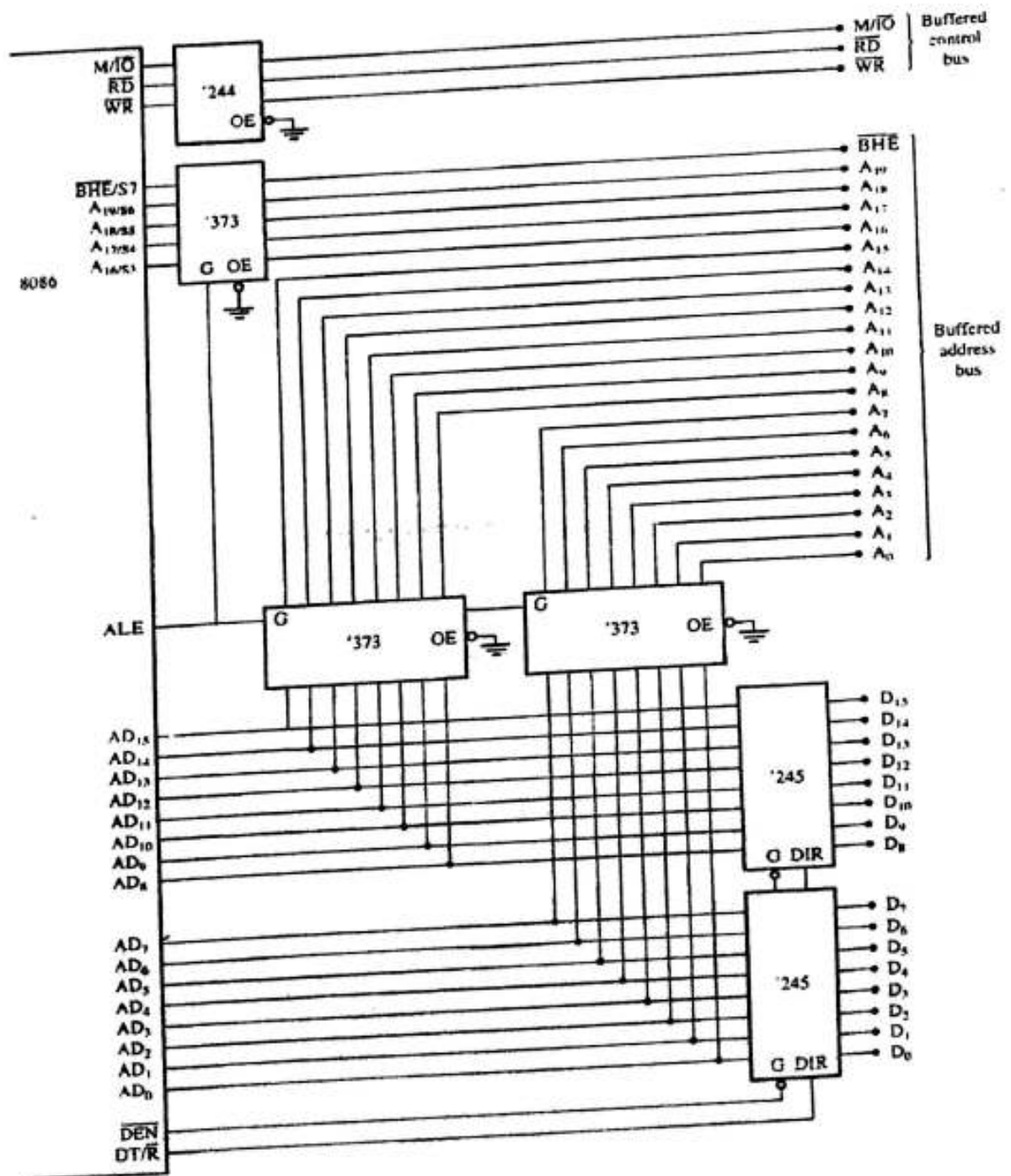
The Fully Buffered 8088. Figure - depicts a fully buffered 8088 microprocessor. Notice that the remaining eight address pins, A_{15} - A_8 , use a 74LS244 octal buffer; the eight data bus pins, D_7 - D_0 , use a 74LS245 octal bi-directional bus buffer; and the control bus signals, $\overline{IO/\overline{M}}$, \overline{RD} , and \overline{WR} , use a 74LS244 buffer. A fully buffered 8088 system requires two 74LS244s, one 74LS245, and two 74LS373s. The direction of the 74LS245 is controlled by the DT/\overline{R} signal, and is enabled and disabled by the \overline{DEN} signal.



A fully buffered 8088 microprocessor

The full buffered 8086

A fully buffered 8086 system requires one 74LS244, two 74LS245s, and three 74LS373s. The 8086 requires one more buffer than the 8088 because of the extra eight data bus connections, $D_{15}-D_8$. It also has a $\overline{\text{BHE}}$ signal that is buffered for memory-bank selection.



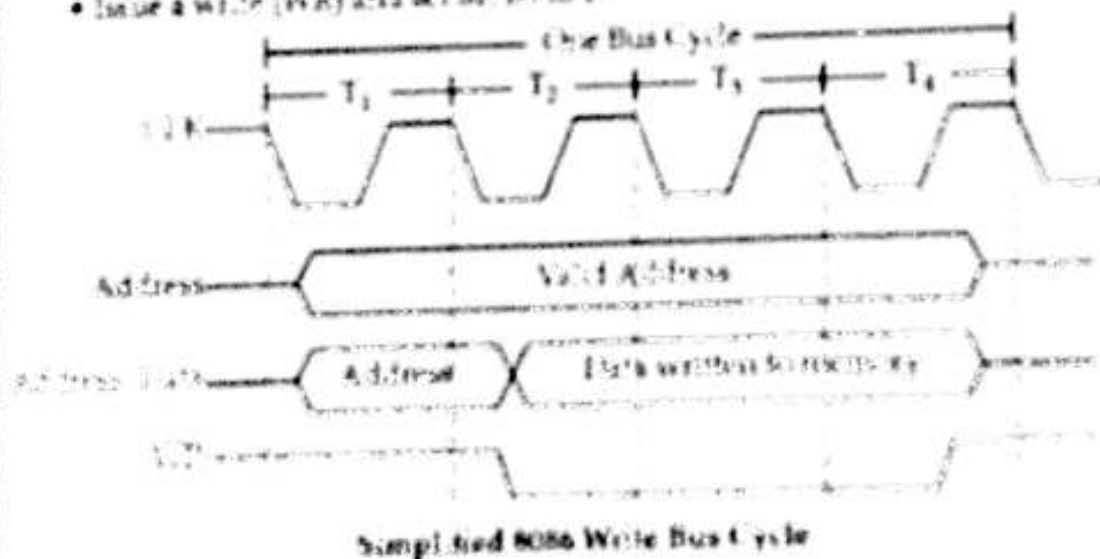
BUS TIMING

- The three buses of the 8086 and 8088 address, data and control function in exactly the same manner as those of any other microprocessor.
- If **data are written** to the memory, the microprocessor outputs the memory address on the address bus, output the data to be written into memory on the data bus, and issues a write (\overline{WR}) to memory and if ($\overline{M/\overline{IO}}$) for the 8088 and $\overline{IO/\overline{M}}=1$ for the 8086.
- If **data are reads** from the memory, the microprocessor outputs the memory address on the address bus, issues a read (\overline{RD}) memory signal, and accepts the data via the data bus.

BUS Timing

Writing

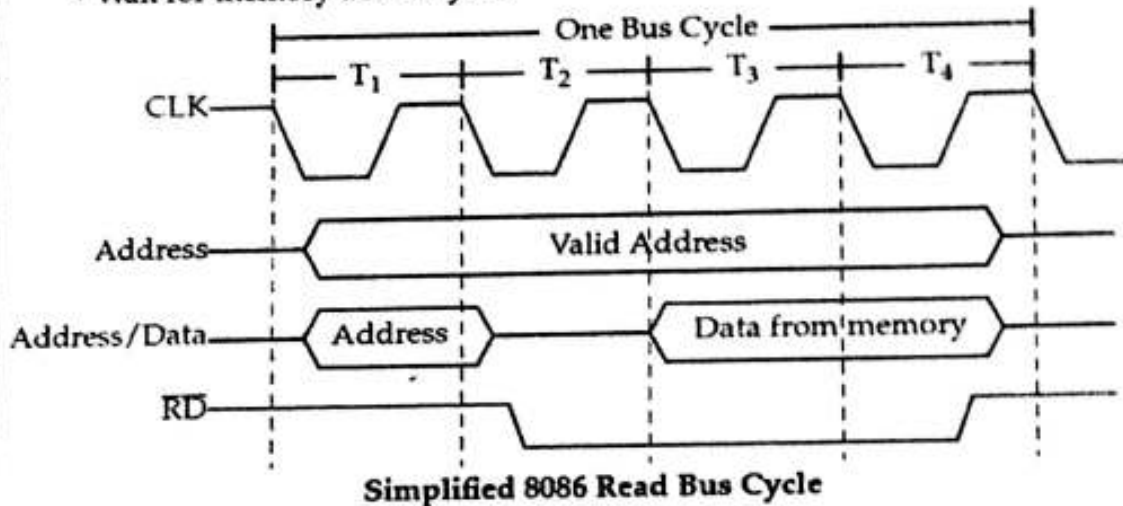
- Dump address on address bus
- Dump data on data bus
- Issue a write (\overline{WR}) and set $\overline{M/\overline{IO}}$ to 1.



BUS Timing

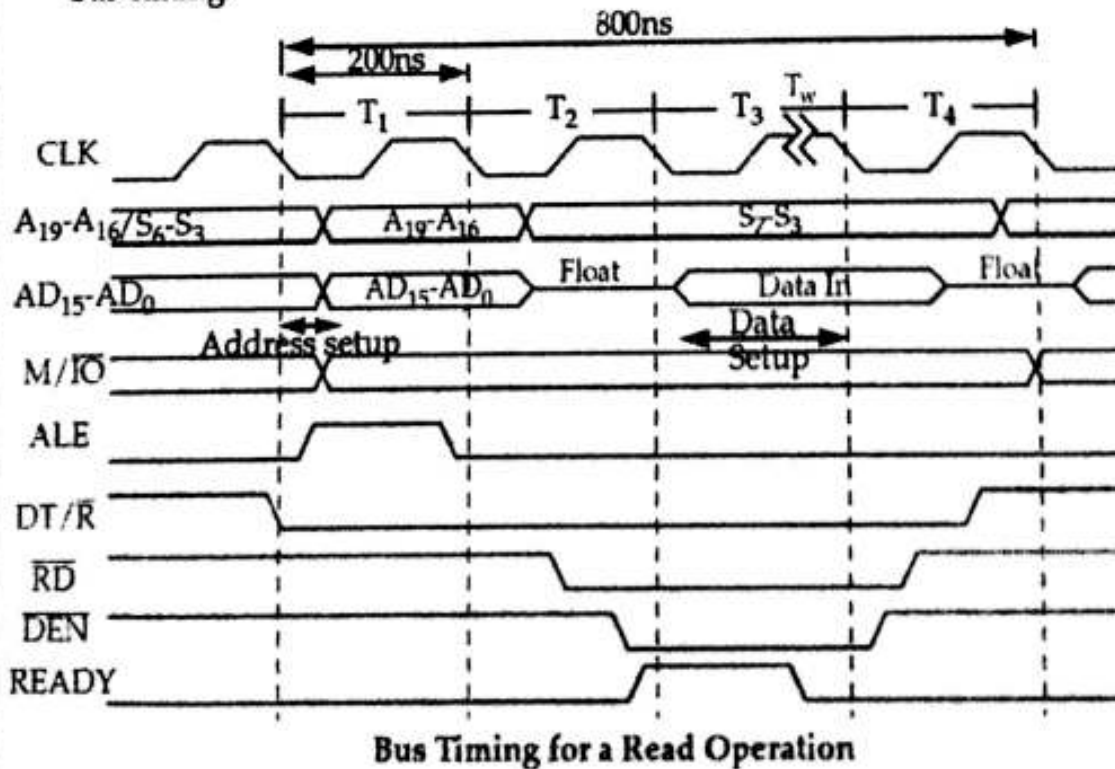
Reading:

- Dump address on address bus.
- Issue a read (\overline{RD}) and set M/\overline{IO} to 1.
- Wait for memory access cycle.



BUS Timing

Bus Timing:



BUS Timing

During T_1 :

- The address is placed on the Address/Data bus.
- Control signals M/\overline{IO} , $A16$ and DT/\overline{R} specify memory or I/O, latch the address onto the address bus and set the direction of data transfer on data bus.

During T_2 :

- 8086 issues the \overline{RD} or \overline{WR} signal, \overline{DEN} , and, for a write, the data. \overline{DEN} enables the memory or I/O device to receive the data for writes and the 8086 to receive the data for reads.

During T_3 :

- This cycle is provided to allow memory to access data.
- \overline{READY} is sampled at the end of T_2 .

If low, T_3 becomes a wait state.

Otherwise, the data bus is sampled at the end of T_3 .

During T_4 :

- All bus signals are deactivated, in preparation for next bus cycle.
- Data is sampled for reads, writes occur for writes.



BUS Timing

Timing:

Each BUS CYCLE on the 8086 equals four system clocking periods (T states).

The clock rate is 5MHz, therefore one Bus Cycle is 800ns.

The transfer rate is 1.25MHz.

Memory specs (memory access time) must match constraints of system timing.

For example, bus timing for a read operation shows almost 600ns are needed to read data.

However, memory must access faster due to setup times, e.g.

Address setup and data setup.

This subtracts off about 150ns.

Therefore, memory must access in at least 450ns minus another 30-40ns guard band for buffers and decoders.

420ns DRAM required for the 8086.

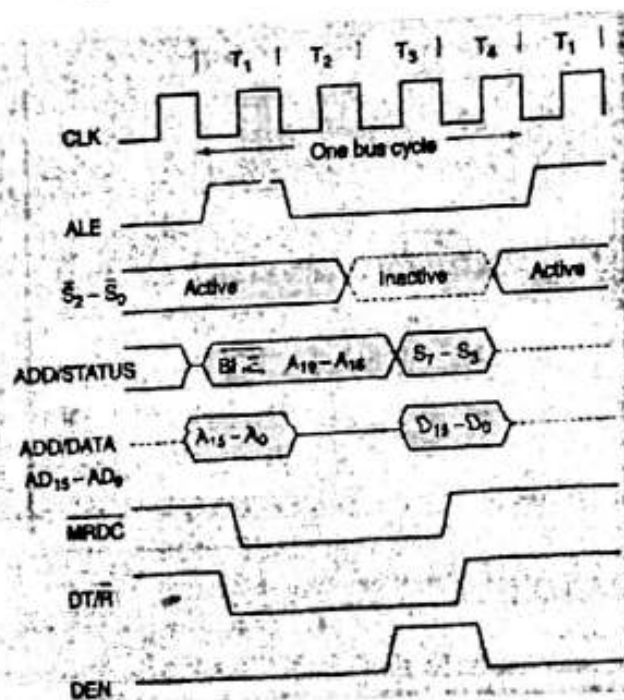
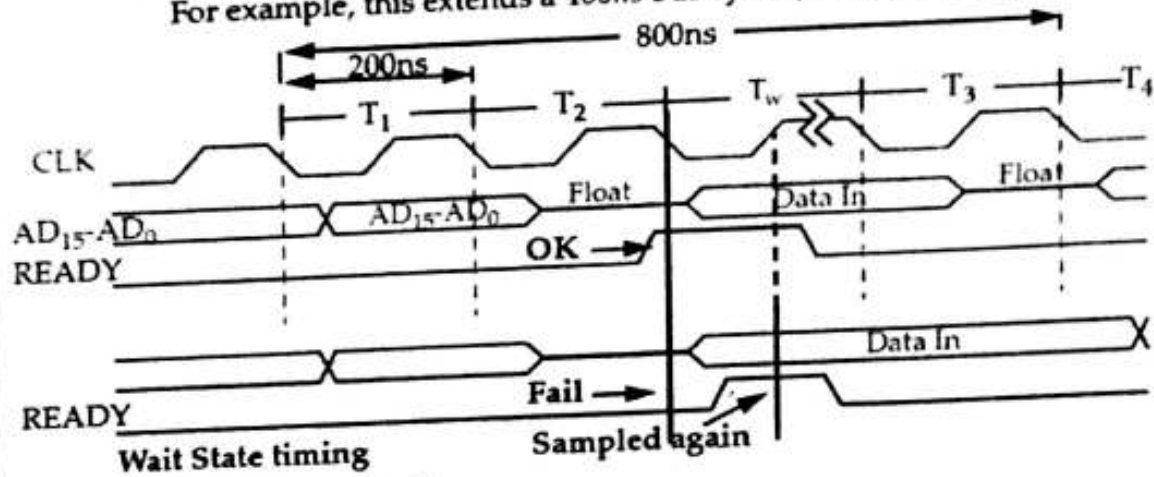
BUS Timing

READY:

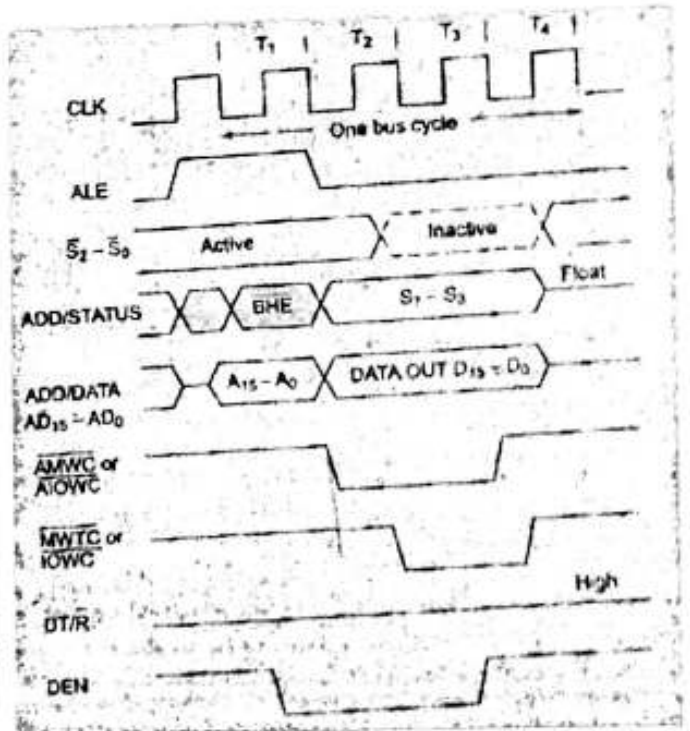
An input to the 8086 that causes wait states for slower memory and I/O components.

A wait state (T_w) is an extra clock period inserted between T_2 and T_3 to lengthen the bus cycle.

For example, this extends a 460ns bus cycle (at 5MHz clock) to 660ns.



Memory Read Timing in Maximum Mode



Memory Write Timing in Maximum Mode

intel.

8086 16-BIT HMOS MICROPROCESSOR 8086/8086-2/8086-1

- Direct Addressing Capability 1 MByte of Memory
- Architecture Designed for Powerful Assembly Language and Efficient High Level Languages
- 14 Word, by 16-Bit Register Set with Symmetrical Operations
- 24 Operand Addressing Modes
- Bit, Byte, Word, and Block Operations
- 8 and 16-Bit Signed and Unsigned Arithmetic in Binary or Decimal Including Multiply and Divide
- Range of Clock Rates:
5 MHz for 8086,
8 MHz for 8086-2,
10 MHz for 8086-1
- MULTIBUS System Compatible Interface
- Available in EXPRESS
— Standard Temperature Range
— Extended Temperature Range
- Available in 40-Lead Cerdip and Plastic Package
(See Packaging Spec. Order # 231360)

The Intel 8086 high performance 16-bit CPU is available in three clock rates: 5, 8 and 10 MHz. The CPU is implemented in N-Channel, depletion load, silicon gate technology (HMOS-III), and packaged in a 40-pin Cerdip or plastic package. The 8086 operates in both single processor and multiple processor configurations to achieve high performance levels.

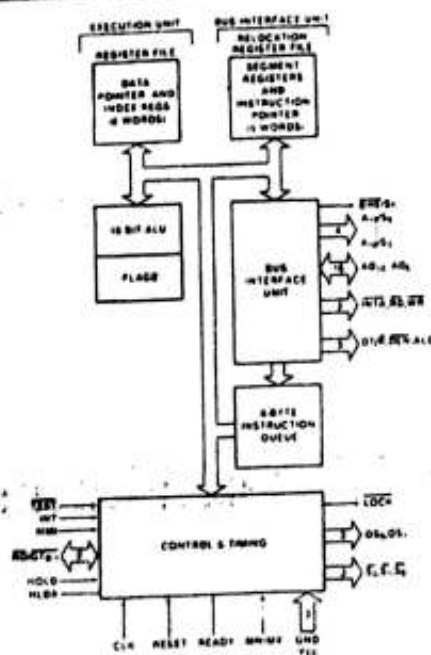


Figure 1. 8086 CPU Block Diagram

231455-1



40 Lead
Figure 2. 8086 Pin Configuration

Order Number: 231455-005

September 1980

Table 1. Pin Description

The following pin function descriptions are for 8086 systems in either minimum or maximum mode. The "Local Bus" in these descriptions is the direct multiplexed bus interface connection to the 8086 (without regard to additional bus buffers).

bus" in these descriptions (the use of additional bus buffers).

Symbol	Pin No.	Type	Name and Function																		
AD ₁₅ -AD ₀	2-16, 39	I/O	ADDRESS DATA BUS: These lines constitute the time multiplexed memory/I/O address (T ₁), and data (T ₂ , T ₃ , T _W , T ₄) bus. A ₀ is analogous to BHE for the lower byte of the data bus, pins D ₇ -D ₀ . It is LOW during T ₁ when a byte is to be transferred on the lower portion of the bus in memory or I/O operations. Eight-bit oriented devices tied to the lower half would normally use A ₀ to condition chip select functions. (See BHE.) These lines are active HIGH and float to 3-state OFF during interrupt acknowledge and local bus "hold acknowledge".																		
A ₁₉ /S ₆ , A ₁₈ /S ₅ , A ₁₇ /S ₄ , A ₁₆ /S ₃	35-38	O	ADDRESS/STATUS: During T ₁ these are the four most significant address lines for memory operations. During I/O operations these lines are LOW. During memory and I/O operations, status information is available on these lines during T ₂ , T ₃ , T _W , T ₄ . The status of the interrupt enable FLAG bit (S ₅) is updated at the beginning of each CLK cycle. A ₁₇ /S ₄ and A ₁₆ /S ₃ are encoded as shown. This information indicates which relocation register is presently being used for data accessing. These lines float to 3-state OFF during local bus "hold acknowledge". <table><tr><th>A₁₇/S₄</th><th>A₁₆/S₃</th><th>Characteristics</th></tr><tr><td>0 (LOW)</td><td>0</td><td>Alternate Data</td></tr><tr><td>0</td><td>1</td><td>Stack</td></tr><tr><td>1 (HIGH)</td><td>0</td><td>Code or None</td></tr><tr><td>1</td><td>1</td><td>Data</td></tr><tr><td>S₆ is 0 (LOW)</td><td></td><td></td></tr></table>	A ₁₇ /S ₄	A ₁₆ /S ₃	Characteristics	0 (LOW)	0	Alternate Data	0	1	Stack	1 (HIGH)	0	Code or None	1	1	Data	S ₆ is 0 (LOW)		
A ₁₇ /S ₄	A ₁₆ /S ₃	Characteristics																			
0 (LOW)	0	Alternate Data																			
0	1	Stack																			
1 (HIGH)	0	Code or None																			
1	1	Data																			
S ₆ is 0 (LOW)																					
BHE/S ₇	34	O	BUS HIGH ENABLE/STATUS: During T ₁ the bus high enable signal (BHE) should be used to enable data onto the most significant half of the data bus, pins D ₁₅ -D ₈ . Eight-bit oriented devices tied to the upper half of the bus would normally use BHE to condition chip select functions. BHE is LOW during T ₁ for read, write, and interrupt acknowledge cycles when a byte is to be transferred on the high portion of the bus. The S ₇ status information is available during T ₂ , T ₃ , and T ₄ . The signal is active LOW, and floats to 3-state OFF in "hold". It is LOW during T ₁ for the first interrupt acknowledge cycle. <table><tr><th>BHE</th><th>A₀</th><th>Characteristics</th></tr><tr><td>0</td><td>0</td><td>Whole word</td></tr><tr><td>0</td><td>1</td><td>Upper byte from/to odd address</td></tr><tr><td>1</td><td>0</td><td>Lower byte from/to even address</td></tr><tr><td>1</td><td>1</td><td>None</td></tr></table>	BHE	A ₀	Characteristics	0	0	Whole word	0	1	Upper byte from/to odd address	1	0	Lower byte from/to even address	1	1	None			
BHE	A ₀	Characteristics																			
0	0	Whole word																			
0	1	Upper byte from/to odd address																			
1	0	Lower byte from/to even address																			
1	1	None																			
RD	32	O	READ: Read strobe indicates that the processor is performing a memory or I/O read cycle, depending on the state of the S ₂ pin. This signal is used to read devices which reside on the 8086 local bus. RD is active LOW during T ₂ , T ₃ and T _W of any read cycle, and is guaranteed to remain HIGH in T ₂ until the 8086 local bus has floated. This signal floats to 3-state OFF in "hold acknowledge".																		

Table 1. Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
READY	22	I	READY: is the acknowledgement from the addressed memory or I/O device that it will complete the data transfer. The READY signal from memory/I/O is synchronized by the 8254A Clock Generator to form READY. This signal is active HIGH. The 8086 READY input is not synchronized. Correct operation is not guaranteed if the setup and hold times are not met.
INTR	18	I	INTERRUPT REQUEST: is a level triggered input which is sampled during the last clock cycle of each instruction to determine if the processor should enter into an interrupt acknowledge operation. A subroutine is vectored to via an interrupt vector lookup table located in system memory. It can be internally masked by software resetting the interrupt enable bit. INTR is internally synchronized. This signal is active HIGH.
TEST	23	I	TEST: input is examined by the "Wait" instruction. If the TEST input is LOW execution continues, otherwise the processor waits in an "Idle" state. This input is synchronized internally during each clock cycle on the leading edge of CLK.
NMI	17	I	NON-MASKABLE INTERRUPT: an edge triggered input which causes a type 2 interrupt. A subroutine is vectored to via an interrupt vector lookup table located in system memory. NMI is not maskable internally by software. A transition from LOW to HIGH initiates the interrupt at the end of the current instruction. This input is internally synchronized.
RESET	21	I	RESET: causes the processor to immediately terminate its present activity. The signal must be active HIGH for at least four clock cycles. It restarts execution, as described in the Instruction Set description, when RESET returns LOW. RESET is internally synchronized.
CLK	19	I	CLOCK: provides the basic timing for the processor and bus controller. It is asymmetric with a 33% duty cycle to provide optimized internal timing.
V _{CC}	40		V _{CC} : +5V power supply pin.
GND	1, 20		GROUND
MN/ \overline{MX}	33	I.	MINIMUM/MAXIMUM: indicates what mode the processor is to operate in. The two modes are discussed in the following sections.

The following pin function descriptions are for the 8086/8288 system in maximum mode (i.e., MN/ \overline{MX} = V_{SS}). Only the pin functions which are unique to maximum mode are described; all other pin functions are as described above.

$\overline{S_2}, \overline{S_1}, \overline{S_0}$	26-28	O	STATUS: active during T ₄ , T ₁ , and T ₂ and is returned to the passive state (1, 1, 1) during T ₃ or during T _W when READY is HIGH. This status is used by the 8288 Bus Controller to generate all memory and I/O access control signals. Any change by $\overline{S_2}$, $\overline{S_1}$, or $\overline{S_0}$ during T ₄ is used to indicate the beginning of a bus cycle, and the return to the passive state in T ₃ or T _W is used to indicate the end of a bus cycle.
--	-------	---	---

Table 1. Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function																																				
S_2, S_1, S_0 (Continued)	26-28	O	<p>These signals float to 3-state OFF in "hold acknowledge". These status lines are encoded as shown.</p> <table> <tr> <th>S_2</th><th>S_1</th><th>S_0</th><th>Characteristics</th></tr> <tr> <td>0 (LOW)</td><td>0</td><td>0</td><td>Interrupt Acknowledge</td></tr> <tr> <td>0</td><td>0</td><td>1</td><td>Read I/O Port</td></tr> <tr> <td>0</td><td>1</td><td>0</td><td>Write I/O Port</td></tr> <tr> <td>0</td><td>1</td><td>1</td><td>Halt</td></tr> <tr> <td>0</td><td>0</td><td>0</td><td>Code Access</td></tr> <tr> <td>1 (HIGH)</td><td>0</td><td>1</td><td>Read Memory</td></tr> <tr> <td>1</td><td>1</td><td>0</td><td>Write Memory</td></tr> <tr> <td>1</td><td>1</td><td>1</td><td>Passive</td></tr> </table>	S_2	S_1	S_0	Characteristics	0 (LOW)	0	0	Interrupt Acknowledge	0	0	1	Read I/O Port	0	1	0	Write I/O Port	0	1	1	Halt	0	0	0	Code Access	1 (HIGH)	0	1	Read Memory	1	1	0	Write Memory	1	1	1	Passive
S_2	S_1	S_0	Characteristics																																				
0 (LOW)	0	0	Interrupt Acknowledge																																				
0	0	1	Read I/O Port																																				
0	1	0	Write I/O Port																																				
0	1	1	Halt																																				
0	0	0	Code Access																																				
1 (HIGH)	0	1	Read Memory																																				
1	1	0	Write Memory																																				
1	1	1	Passive																																				
RQ/GT_0 RQ/GT_1	30, 31	I/O	<p>REQUEST/GRANT: pins are used by other local bus masters to force the processor to release the local bus at the end of the processor's current bus cycle. Each pin is bidirectional with RQ/GT_0 having higher priority than RQ/GT_1. RQ/GT pins have internal pull-up resistors and may be left unconnected. The request/grant sequence is as follows (see Page 2-24):</p> <ol style="list-style-type: none"> 1. A pulse of 1 CLK wide from another local bus master indicates a local bus request ("hold") to the 8086 (pulse 1). 2. During a T_4 or T_1 clock cycle, a pulse 1 CLK wide from the 8086 to the requesting master (pulse 2), indicates that the 8086 has allowed the local bus to float and that it will enter the "hold acknowledge" state at the next CLK. The CPU's bus interface unit is disconnected logically from the local bus during "hold acknowledge". 3. A pulse 1 CLK wide from the requesting master indicates to the 8086 (pulse 3) that the "hold" request is about to end and that the 8086 can reclaim the local bus at the next CLK. <p>Each master-master exchange of the local bus is a sequence of 3 pulses. There must be one dead CLK cycle after each bus exchange. Pulses are active LOW.</p> <p>If the request is made while the CPU is performing a memory cycle, it will release the local bus during T_4 of the cycle when all the following conditions are met:</p> <ol style="list-style-type: none"> 1. Request occurs on or before T_2. 2. Current cycle is not the low byte of a word (on an odd address). 3. Current cycle is not the first acknowledge of an interrupt acknowledge sequence. 4. A locked instruction is not currently executing. <p>If the local bus is idle when the request is made the two possible events will follow:</p> <ol style="list-style-type: none"> 1. Local bus will be released during the next clock. 2. A memory cycle will start within 3 clocks. Now the four rules for a currently active memory cycle apply with condition number 1 already satisfied. 																																				
\overline{LOCK}	29	O	<p>\overline{LOCK}: output indicates that other system bus masters are not to gain control of the system bus while \overline{LOCK} is active LOW. The \overline{LOCK} signal is activated by the "LOCK" prefix instruction and remains active until the completion of the next instruction. This signal is active LOW, and floats to 3-state OFF in "hold acknowledge".</p>																																				

Table 1. Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function		
QS ₁ , QS ₀	24, 25	O	QUEUE STATUS: The queue status is valid during the CLK cycle after which the queue operation is performed. QS ₁ and QS ₀ provide status to allow external tracking of the internal 8086 instruction queue.		
			QS ₁	QS ₀	Characteristics
			0 (LOW)	0	No Operation
			0	1	First Byte of Op Code from Queue
			1 (HIGH)	0	Empty the Queue
			1	1	Subsequent Byte from Queue

The following pin function descriptions are for the 8086 in minimum mode (i.e., $MN/\overline{MX} = V_{CC}$). Only the pin functions which are unique to minimum mode are described; all other pin functions are as described above.

M/ \overline{IO}	28	O	STATUS LINE: logically equivalent to S ₂ in the maximum mode. It is used to distinguish a memory access from an I/O access. M/ \overline{IO} becomes valid in the T ₄ preceding a bus cycle and remains valid until the final T ₄ of the cycle (M = HIGH, IO = LOW). M/ \overline{IO} floats to 3-state OFF in local bus "hold acknowledge".
WR	29	O	WRITE: indicates that the processor is performing a write memory or write I/O cycle, depending on the state of the M/ \overline{IO} signal. WR is active for T ₂ , T ₃ and T _W of any write cycle. It is active LOW, and floats to 3-state OFF in local bus "hold acknowledge".
INTA	24	O	INTA: is used as a read strobe for interrupt acknowledge cycles. It is active LOW during T ₂ , T ₃ and T _W of each interrupt acknowledge cycle.
ALE	25	O	ADDRESS LATCH ENABLE: provided by the processor to latch the address into the 8282/8283 address latch. It is a HIGH pulse active during T ₁ of any bus cycle. Note that ALE is never floated.
DT/ \overline{R}	27	O	DATA TRANSMIT/RECEIVE: needed in minimum system that desires to use an 8286/8287 data bus transceiver. It is used to control the direction of data flow through the transceiver. Logically DT/ \overline{R} is equivalent to S ₁ in the maximum mode, and its timing is the same as for M/ \overline{IO} . (T = HIGH, R = LOW.) This signal floats to 3-state OFF in local bus "hold acknowledge".
DEN	26	O	DATA ENABLE: provided as an output enable for the 8286/8287 in a minimum system which uses the transceiver. DEN is active LOW during each memory and I/O access and for INTA cycles. For a read or INTA cycle it is active from the middle of T ₂ until the middle of T ₄ , while for a write cycle it is active from the beginning of T ₂ until the middle of T ₄ . DEN floats to 3-state OFF in local bus "hold acknowledge".
HOLD, HLDA	31, 30	I/O	HOLD: indicates that another master is requesting a local bus "hold." To be acknowledged, HOLD must be active HIGH. The processor receiving the "hold" request will issue HLDA (HIGH) as an acknowledgement in the middle of a T ₄ or T ₁ clock cycle. Simultaneous with the issuance of HLDA the processor will float the local bus and control lines. After HOLD is detected as being LOW, the processor will LOWER the HLDA, and when the processor needs to run another cycle, it will again drive the local bus and control lines. Hold acknowledge (HLDA) and HOLD have internal pull-up resistors. The same rules as for $\overline{RD}/\overline{GT}$ apply regarding when the local bus will be released. HOLD is not an asynchronous input. External synchronization should be provided if the system cannot otherwise guarantee the setup time.

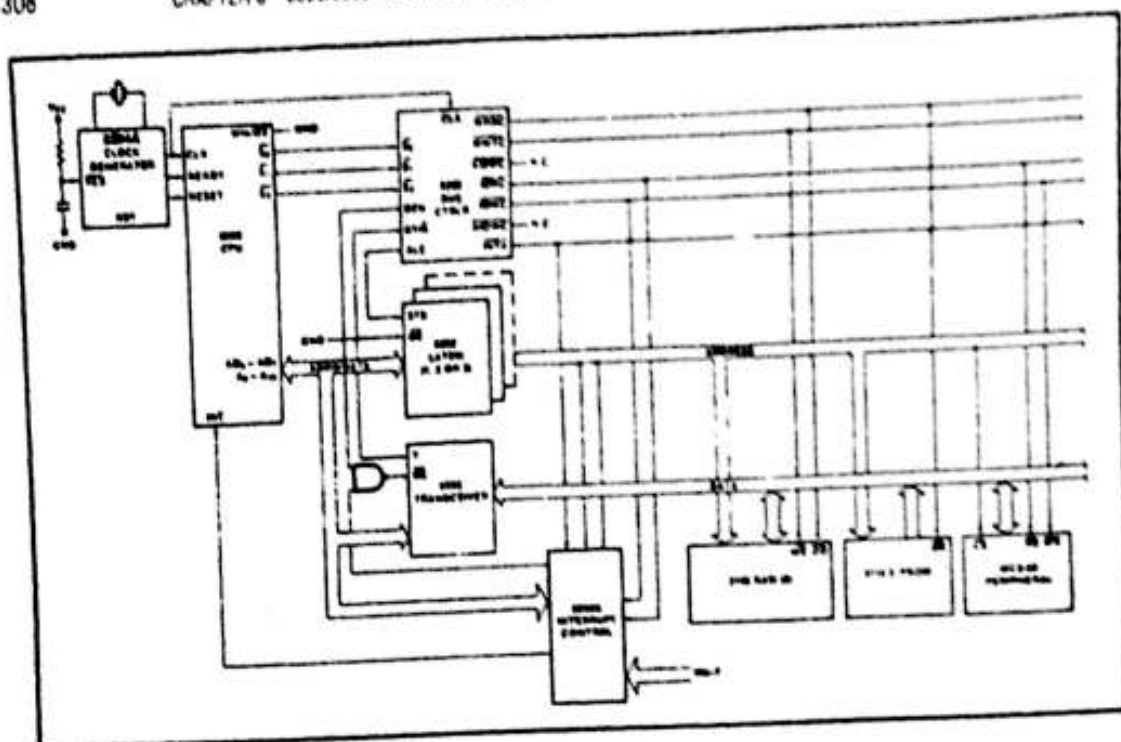


FIGURE 8-20 Maximum mode 8088 system

the 8086/8088 for bus control during maximum mode because new pins and new features have replaced some of them. Maximum mode is used only when the system contains external co-processors such as the 8087 arithmetic coprocessor.

The 8288 Bus Controller

An 8086/8088 system that is operated in maximum mode must have an 8288 bus controller to provide the signals that are eliminated from the 8086/8088 by the maximum mode operation. Figure 8-21 illustrates the block diagram and pin-out of the 8288 bus controller circuit.

Notice that the control bus developed by the 8288 bus controller contains separate signals for I/O (\overline{IORC} and \overline{IOWC}) and memory (\overline{MRDC} and \overline{MWTC}). It also contains advanced memory (\overline{AMWC}) and I/O (\overline{AIOWC}) write strobes and the \overline{INTA} signal. These signals replace the minimum mode \overline{ALE} , \overline{WR} , $\overline{IO/M}$, $\overline{DT/R}$, \overline{DEN} , and \overline{INTA} , which are lost when the 8086/8088 is operated in the maximum mode.

Pin Functions. The following list provides a description of each pin of the 8288 bus controller.

- | | |
|-----------------------|---|
| S2, S1, and S0 | Status inputs are connected to the status output pins on the 8086/8088 microprocessors. These three signals are decoded to generate the timing signals for the system. |
| CLK | The clock input provides internal timing and must be connected to the CLK output pin of the 8284A clock generator. |
| ALE | The address latch enable output is used to demultiplex the address/data bus. |
| DEN | The data bus enable pin controls the bi-directional data bus buffers in the system. Note that this is an active high-output pin that is the opposite polarity from the \overline{DEN} signal found on the microprocessor when operated in the minimum mode. |

8-7 SUMMARY

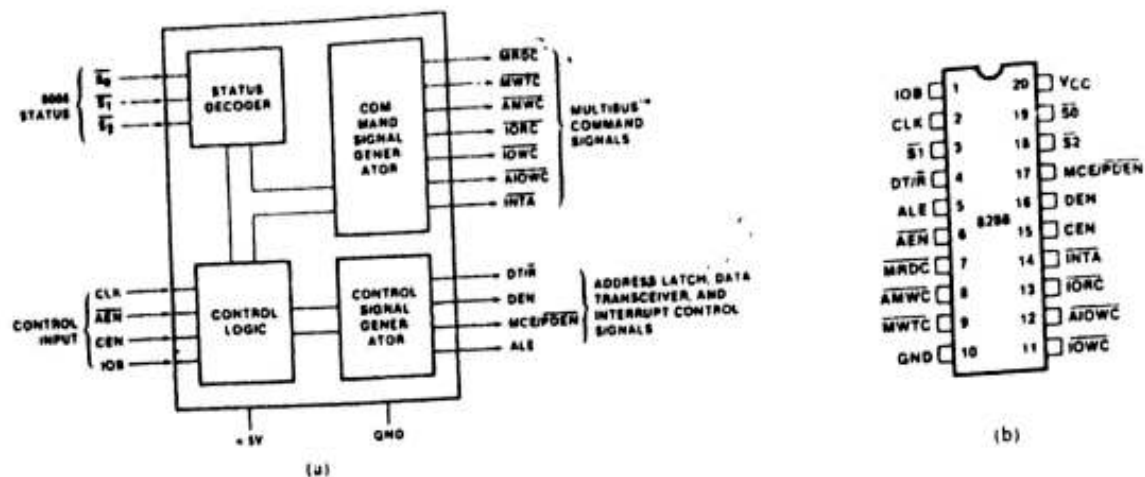


FIGURE 8-21 The 8288 bus controller. (a) block diagram (b) pinout

 $\overline{DT/R}$

The data transmit/receive signal is output by the 8288 to control the direction of the bi-directional data bus buffers.

 \overline{AEN}

The address enable input causes the 8288 to enable the memory control signals.

 \overline{CEN}

The control enable input enables the command output pins on the 8288.

 \overline{IOB}

The I/O bus mode input selects either the I/O bus mode or system bus mode operation.

 \overline{AIOWC}

The advanced I/O write command output provides I/O with an advanced I/O write control signal.

 \overline{IOWC}

The I/O write command output provides I/O with its main write signal.

 \overline{IORC}

The I/O read command output provides I/O with its read control signal.

 \overline{AMWC}

The advanced memory write control pin provides memory with an early or advanced write signal.

 \overline{MWTC}

The memory write control pin provides memory with its normal write control signal.

 \overline{MRDC}

The memory read control pin provides memory with a read control signal.

 \overline{INTA}

The interrupt acknowledge output acknowledges an interrupt request input applied to the \overline{INTR} pin.

 $\overline{MCE/PDEN}$

- The master cascade/peripheral data output selects cascade operation for an interrupt controller if \overline{IOB} is grounded and enables the I/O bus transceivers if \overline{IOB} is tied high.

8-7

SUMMARY

1. The main differences between the 8086 and 8088 are (a) an 8-bit data bus on the 8088 and a 16-bit data bus on the 8086, (b) an \overline{SSO} pin on the 8088 in place of $\overline{BHE/S_7}$ on the 8086, and (c) an $\overline{IO/\overline{M}}$ pin on the 8088 instead of an $\overline{M/\overline{IO}}$ on the 8086.